$1.50

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

November/December, 1976 Box E, Menlo Park CA 94025 Volume 1, Number 10

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

– – and dozens of other tidbits – –

DON'T KEEP IT A SECRET!

Let us know what exciting new software and systems you are working on. We'll tell everyone else (if you wish). Maybe someone is also working on the same thing. You can work together and get results twice as fast. Or, may be someone else has already done it; no reason for everyone to reinvent the wheel.

DR. DOBB'S JOURNAL OF
COMPUTER CALISTHENICS & ORTHODONTIA

Volume 1, Number 10, November – December, 1976
Box E, Menlo Park, CA 94025

Publisher
People's Computer Company
1010 Doyle, Menlo Park, California
(415) 323-3111
Editor
Jim C. Warren, Jr.
Contributing Editors
Marvin Winzenread
Jim Day
Product & Software Evaluation Group
Jef Raskin, Director
Michael Heathman
Dennis McGhie
Watchdogs
Bob Albrecht
Dennis Allison
Underdog
Rosehips Malloy
Circulation & Subscriptions
R. Jacobsen
Bulk Sales
Dan Rosset

POSTMASTER: *Please send form 3579 to: Box E, Menlo Park, CA 94025. Return postage guaranteed. Second-class postage paid at Menlo Park, CA. Published 10 times per year, excluding July & December.*

U.S. Subscriptions
☐ $12 for one year
☐ $22 for two years
☐ $21 for one year for first class/airmail to anywhere in U.S.

Back Issues to U.S.
☐ All of Vol. 1 for $13
Vol. ___ No. ___ for $1.50
Vol. ___ No. ___ for $1.50
Vol. ___ No. ___ for $1.50

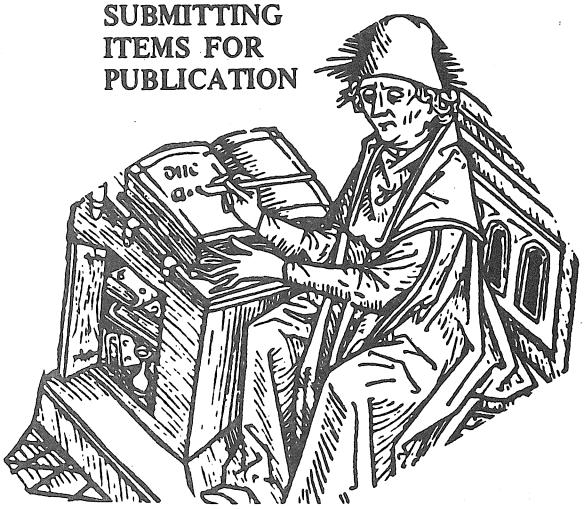
Foreign via Surface Mail
☐ $16/year – anywhere outside U.S.
(2nd-Class Regulations require surcharge on mail sent to Canada)

Foreign Via Air Mail
☐ $21/year to Canada
☐ $28/year to Europe & Pan America
☐ $32/year – other foreign

European Distributor:
Pan Atlantic Computer Systems, gmbh
Frankfurter Strasse 78
D61 Darmstadt, West Germany

European Rates:
35 DM/year in West Germany
£ 8.00/year in Britain

SUBMITTING ITEMS FOR PUBLICATION



DATE'M—Please include your name, address, and *date* on all tidbits you send to us.

TYPE'M—If at all possible, items should be typewritten, double-spaced, on standard, 8½ x 11 inch, white paper. If we can't read it, we can't publish it. Remember that we will be retyping all natural language (as opposed to computer languages) communications that we publish.

PROGRAM LISTINGS—We will accept hand-written programs *only as a very last resort.* Too often, they tend to say something that the computer would find indigestible. On the other hand, if the computer typed it, the computer would probably accept it—particularly if it is a listing pass from an assembler or other translator.

It is significantly helpful for program listings to be on continuous paper; either white, or very light blue, roll paper, or fan-folded paper. Since we reduce the copy in size, submitting it on individual pages forces us to do a significant amount of extra cutting and pasting. For the same reason, we prefer that you *exclude* pagination or page headings from any listings.

Please, please, please put a new ribbon on your printer before you run off a listing for publication.

In any natural language documentation accompanying a program listing, please refer to portions of code by their address or line number or label, rather than by page number.

DRAWINGS & SCHEMATICS—Please draw them significantly larger than the size you expect them to be when they are published. Take your time and make them as neat as possible. We do not have the staff to retouch or re-draw illustrations. Use a black-ink pen on white paper.

LETTERS FOR PUBLICATION—We are always interested in hearing your praise, complaints, opinions, daydreams, etc. In letters of opinion for publication, however, please back up any opinions that you present with as much factual information as possible.

We are quite interested in publishing well-founded, responsible evaluations and critiques of anything concerning hobbyist hardware or software, home computers, or computers and people.

We may withhold your name from a published letter if you so request. We will not publish correspondence, however, which is sent to us anonymously.

We reserve the right to edit letters for purpose of clarity and brevity.

ADVERTISING—As long as we can afford to do so, we will not accept paid commercial advertising. This "keeps us honest" when we pursue the role of consumer advocate.

CONSUMER ACTION

PRODUCT & SOFTWARE EVALUATION AND TESTING TO BECOME REGULAR FEATURE IN *DR. DOBB'S JOURNAL*

With this issue, we are initiating what we expect to be a regular feature in *Dr. Dobb's Journal:* reports of independent product and software tests and evaluations. We propose that these will be "independent" in that we have no financial ties or obligations to these manufacturers, the producers of these products. We carry no paid advertising.

Ever since the computer hobby began, there have been regular pleas for such independent testing and evaluation. Until recently, we have been rather haphazard in our attempts to assist consumers in judging the quality of products being marketed to them. We have pursued this primarily through the publication of complimentary and complaining letters regarding products. With such letters, we generally have no knowledge of the expertise, fairness, honesty or bias of the writers (thus, they have been published as "letters" rather than as "articles"). Recognition of this fact prompted us to adopt a policy [see Editorial in October, 1976, *DDJ*] regarding the treatment of letters of complaint. Though we will continue to publish such letters within the constraints of that policy [see several examples in this issue], we feel that a formal, orderly product testing and evaluation program would be more fair and more useful to our readers. It will also be perfectly in keeping with the Charter of our publisher, People's Computer Company. PCC is a California-licensed, non-profit educational corporation.

WHO WILL DO THE TESTING?

We have organized an evaluation team consisting of three people, plus the Editor. These are individuals whose qualifications we *do* know. Jef Raskin is the Director of the group. Many of you already know of him through his critique of a number of hobby systems [*DDJ*, September, 1976, "A Bit of Wheat Amongst the Chaff." This issue carries a second product evaluation by him. Jef is currently an independent consultant involved in several "real world" applications of small computers. Prior to this, his work included serving as Director of the Third College Computation Center at the University of California in San Diego, and serving as a Professor of Visual Arts there for five years. Before that, he was an instructor in Computer and Electronic Music at Pennsylvania State University for several years. He holds a B.S. in Philosophy (1965) from the University of New York with minors in mathematics and physics, and a M.S. in Computer Science (1967) from Penn State.

The second member of the evaluation group is Dennis McGhie. Dennis is currently working for a major biomedical research center in the San Francisco Bay Area. He has been a programmer since 1968, working on both maxi's and mini's, primarily in the areas of database systems, training systems, computer graphics, and real-time systems. Though he has no formal hardware training, he has a good seat-of-the-pants background derived from years of working with experimental real-time computerized biomedical data acquisition and process control. He holds a B.S. (1968) in Chemistry from Stanford University.

The third team member is Michael Heathman, currently a senior systems programmer for a new time-sharing system being installed in a major Bay Area research institution. He has systems experience with maxi's and midi's, including PDP-15's and PDP-11's. He has been a programmer for six years, except for a year's leave taken to perform graduate studies in computer science at the University of Washington. He holds a B.S. (1970) in electrical engineering from Stanford.

Final responsibility for this program will rest with the Editor of *Dr. Dobb's Journal,* Jim Warren. Aside from editing the *Journal,* currently, Jim is working as an independent consultant specializing in small computers in highly interactive environments. He is also in "dissertation mode" in a Ph.D. program through Stanford's Electrical Engineering Department. He has worked as a computer consultant for most of a decade, with several years of programming experience preceeding that. Prior to entering the computer field, he taught mathematics for about ten years, including Chairing the Mathematics Department at the College of Notre Dame – Belmont. He holds a B.S. (1959) and M.A. (1964) in mathematics, and two M.S. degrees; one in Medical Information Science and the other in Computer Engineering.

HOW WILL THE EVALUATIONS BE DONE?

We will contact producers of products being marketed to the computer hobbyist community, and encourage them to participate in this testing and evaluation program. If they choose to do so, this is what will happen:

They will send us a purchase letter – a voucher with which we can "blind purchase" an item to be tested. We will then have someone, not known to be associated with PCC or *DDJ,* obtain the desired product. In the case of products sold only by mail, a unit will be ordered; when it arrives, it will be "paid for" by returning the voucher with the invoice. If the desired unit is available through Bay Area retailers, our "buyer" will go in; pick out a unit; and, when it comes time to pay for the item, will use the voucher to "pay" for it. The voucher, of course, will guarantee to immediately replace the unit or reimburse the dealer. In this way, we can be reasonably assured of obtaining units for testing that have not been especially "tuned"; they will be standard consumer products.

The product thus obtained will then be evaluated in whatever manner is most appropriate (and in whatever ways are possible with the test gear available at the time). In the case of kits – where such independent examinations are perhaps most badly needed – a team member may construct the kit, or we may well have some interested novice put the kit together under our observation. In either case, careful notes will be kept concerning all aspects of the unit and its evaluation.

Subsystems that are advertised as being plug compatible with some particular interface structure will be tested for such compatability. Major components will be tested against manufacturer's advertising claims and the rated capacities given in the documentation. Other testing will be conducted, where appropriate.

When the testing is completed, the evaluator(s) will write a report of their findings – good and bad. Particular attention will be given to reporting the following aspects: does the unit perform as advertised? Are there inadequacies that are unmentioned by the manufacturer? How does the unit compare to its competitors? How does the unit compare against "perfection"? (When comparing against perfection, the report will explicitly point out that no one else's product meets those standards, either.) Are there any "little hidden gotchas"? In all cases, as much of the test data as possible will be provided in the article, from which the reader will be able to judge the unit for himself. Any personal judgements made by the evaluator will be accompanied by the hard data he used in reaching that opinion, and/or will include an explicit statement of the evaluator's personal bias in the matter.

Once completed, the report will be submitted to the manufacturer of the product for their comments. They will have the opportunity to offer corrections of fact (the "factual" character will be judged by the evaluators and the Editor). If they wish, they may also provide a "manufacturer's response" which, if concise and pertinent in the judgement of the Editor, will be published along with the evaluation. Though the manufacturers will have the right to suggest corrections and to provide a response article, they will have no editorial control over the report. After giving the manufacturer several weeks to a month – and no more than that – to respond, the report and any appropriate response will be published in *Dr. Dobb's Journal.*

The products that are tested will become the property of People's Computer Company. Often, the evaluators will be "paid" for their services by giving them the products they have evaluated. This will both assure an active interest in the evaluation on the part of the team members, and will also provide us with the opportunity to observe

the units in operation over some period of time, with supplementary reports being given as appropriate.

WILL THE MANUFACTURERS COOPERATE?

Both the public and the manufacturers have often noted that commerical magazines rarely if ever publish articles that are really critical of products available from companies that are advertising in that magazine. (One publisher, targeting for the computer hobbyist community, is widely reputed to consistently publish excellent "evaluations" of products from advertisers, and highly critical "evaluations" of products from manufacturers who refuse to advertise in the publication).

Thus, the public tends to take such evaluations in ad-carrying periodicals with a well-deserved grain of salt. Alternatively, *Dr. Dobb's Journal* — and *PCC* before it — has had a consistent reputation for readily publishing compliments and complaints, including complaints about our publications. We believe that carefully done, comprehensive product evaluations, published in *DDJ*, will be accepted as being unbiased and accurate.

If we publish a favorable evaluation of a product, it should be of significant value to that manufacturer. In particular, we herewith grant explicit permission to any manufacturer to reprint such evaluations from *Dr. Dobb's Journal*, either in their entirety or a paragraph at a time. Such reprinting may be done without further permission from *DDJ*, and without any compensation whatsoever being paid to *Dr. Dobb's* (other than possession of the products that were tested). Publication of less than a paragraph at a time, minimum, will require explicit permission from *DDJ*. We explicitly prohibit reprinting out-of-context portions of such a (copyrighted) evaluation, when it fails to accurately reflect the results of the evaluation.

We also hope to — in the not too distant future — develop an objective rating procedure for home computing products. Once developed, we will invent and trademark a *DDJ* evaluation logo that includes the ratings. Manufacturers may then mark their products as "Grade A — Tested in Dr. Dobb's Kitchen", or some such thing.

We have already spoken with several manufacturers, outlining our plans. Reception has varied from an active interest in immediately participating, to a total rejection of the proposal. We will report the details, whenever it seems fair and appropriate to do so.

UNRESPONSIVENESS FROM ADVANCED MICRO-ELECTRONICS

I received a AY5-8500 6 Game MOS/LSI chip from Advanced Micro-Electronics, P.O. Box 17329, Irvine, CA 92713. It didn't operate properly. I wrote them a letter in August (two months ago) describing what the chip did and have received no answer.

Thank you for reading my letter.
Stuart R. Fallgatler
7910 Rio Vista Dr.
Goleta, CA 93017

[We wrote them, saying:] 76-11-7

We recently received a complaint concerning your company, a copy of which is enclosed. Recognizing that there are two sides to every story, and in keeping with our published policy (copy enclosed) [See *DDJ*, Vol. 1, No. 9] concerning handling of consumer complaints regarding vendors' products and services, we wish to offer you the opportunity to present your view of the situation. Therefore, we will withhold any decision concerning possible publication of the complaint for *at least two weeks* from the date of this letter, pending the possibility that you may wish to offer a timely response.

If you do choose to respond, we will, of course, take your comments into consideration in deciding whether or not to publish the complaint. If we do decide to publish it, even in light of your comments, we will almost certainly also publish your response — your side of the story — *unless you explicitly prohibit our publication of your reply.*

Also, if you choose to reply, we would appreciate your forwarding a copy of that reply to the complainant. Many thanks for your attention to these comments. We look forward to your reply.

As of Dec. 2, 1977, we have received no reply.—JCW

BEWARE! THE DOLLAR GOBBLING INFLATION INFECTION IS ABOUT TO AFFLICT ONE OF YOUR LOVED ONES - -

Subscription Rates for Dr. Dobb's Journal *Increase January 1st*

Almost all of our subscription rates are going up as of the start of 1977. This means that our basic subscription rate will now be as much as *Byte's*. And, as always, a year's subscription is for 10 issues; we publish single issues for June/July and Nov./Dec.

We had to either do this — and remain responsible only to our readers — or begin accepting paid advertising, along with its strong though perhaps subtle incentive to "keep the advertisers happy." Considering that we have been purusing an active consumer advocacy role, ever since we started, and considering that we are significantly expanding that activity [see editorial on product and software testing and evaluation plans], we felt that the subscription increase was the preferable alternative. We are still awaiting the results of the question concerning whether or not we should carry paid advertising (a question posed in the last several subscription forms and in the subscription renewal notices).

Prize

DDJ SEEKS *SUPER LOGO!*

Like all massive organizations intent upon changing the fabric of society, *Dr. Dobb's Journal* has concluded that it should have a logo — a symbol by which all people may instantly recognize us. It might be our current title masthead . . . but that's *so* longwinded. Ideally, it should be a symbol or figure that in some sense illustrates our activities (now, now — be nice).

Knowing that computer people are delightfully inventive, we are coming to you for suggestions. We are looking for a logo that we can use in fairly large size in our masthead, letterhead stationery, advertisements, etc. We would also like for it to be recognizable, even when shrunken down to, say, 1"x½". Thus, it can't have too much detail in it (or, the large version can have details, and the smaller versions must be in some way simplified).

Please forward your suggestions. You can describe them, or you can provide a rough sketch, or you can submit an oversized camara-ready master. If we pick your suggestion as the basis for our logo, then we will thank you by giving you a five-year subscription to *DDJ* (extending your current subscription, if necessary). Of course, all suggestions become the property of People's Computer Company, the publisher of *Dr. Dobb's*.

NOW *WE* CAN BLAME IT ON THE COMPUTER

At long last, we have switched from manual processing of subscription records — so fraught with human error — to computerized subscription processing — thereby obtaining even *more* potential for human error. Therefore, please check your address label, and let us know if it is in any way incorrect.

It must have been a computer error . . .

CORRECTION TO PHONE NUMBER FOR KENTUCKY FRIED COMPUTERS

Our September issue carried an announcement of a 10% discount on selected products, available to *DDJ* readers for a limited amount of time, offered by Kentucky Fried Computer Store in Berkeley, CA. We included the phone number, only to be told later that it was incorrect. When we checked the original copy submitted by the store owners – a computer-edited article – we found that we had correctly copied an incorrect number. Tsk, tsk . . . must have been the computer.

Their correct phone is (415) 549-0858, and they are located at 2465 Fourth St.

NEW LOGARITHMIC CONVERTER

by Jim Day

Precision Monolithics, Inc., 1500 Space Park Dr., Santa Clara, CA 95050, 408/246-9222, recently announced development of a D/A converter providing the 72-dB dynamic output range of a 12-bit converter from an 8-bit input. Three bits select one of eight chords (i.e., ranged approximating a logarithmic function) and four bits select one of 16 linear steps within each chord. Resolution near zero is equal to that of a 12-bit converter, dropping to 5-bits (plus sign) at the extremes. Designated the DAC-76, this 18-pin DIP costs $19 in lots of 100.

For high-quality audio output having negligible quantization error at low volume levels, 12-bit D/A converters are customarily used. These tend to be expensive and awkward to drive from an 8-bit MPU. Fortunately, the amplitude response of the human ear is logarithmic. This means that greater quantization error is tolerable at high volume, amking an 8-bit logarithmic D/A converter ideal for speech synthesis and computer–generated music when used with an 8-bit MPU.

A SUPER BOOK, FULL OF *COMPLETE* SYSTEMS PROGRAMS

Dear Dr., September 27, 1976

A valuable new book is available for the computer hobbyist. *Software Tools* by B. W. Kernighan and P. J. Plauger, Reading, MA: Addison-Wesley, 1976 presents programs for a test editor, file formatter, macro processor, librarian and language preprocessor while teaching structured programming. These are *complete programs* available from the publisher in machine-readable form (cost unknown) for a machine with a Fortran compiler. I've read the book – it's great! It starts with a simple echoing routine and builds and builds very logically.

A Tiny Fortran compler with integer arithmetic, character I/O, the IF statement and FUNCTION and SUBROUTINE subprograms could implement an impressive array of tools. Tiny BASIC could do it if it were compilable and could pass arguments as parameters to subroutines.

Implementation of these programs would be a big step toward having home computers help their owners do useful things; and home microcomputers are admirably suited to the word processing tasks the book presents.

It's sort of cheap, too: $8.95 in paper.

Bill Pearson Division of Biology 156-29
Calif. Instit. of Technology Pasadena, CA 91125

Posted 76-10-25

SCCS INTERFACE – STATUS REPORT

Good news! Your regular *SCCS Interface* will be coming again to you soon! This is to bring you further up to date on the Southern California Computer Society publication.

Originally the Society went to an outside service to print *SCCS Interface* on behalf of the Society. Certain differences have arisen with the publishing service and efforts at settlement have apparently failed. During our negotiations, the outside service printed its own magazine called *Interface Age*, the first copy of which appeared in August. You may have received copies of *Interface Age* in the mail. The Society did *not* mail it to its members. The logotype on *Interface Age* and the format of the magazine are very similar to *SCCS Interface* and you may not have even noticed the change. *Interface Age* is *not* an authorized publication of the Society. We have discussed our legal options with our attorneys. Now that we are free to move ahead, the Society has obtained its own publishing service. We will resume distribution of *SCCS Interface* next month. Only *SCCS Interface* will be the authorized publication of the Southern California Computer Society.

We expect *SCCS Interface* will carry out the spirit and policy of the Society – to be objective with regard to vendors' products and services, to report the activities of our Society, to provide an open forum for our members, to experiment and of course, to provide important articles of interest.

We are working hard, fast and enthusiastically on this and appreciate your patience. The memberships of those who missed any copies will be extended.

Larry Press has been named to fill the editor's spot. Please send editorial contributions and suggestions, articles, announcements, inquiries on ads or distribution, aspirin and good wishes to Larry at 1702 Ashland Ave., Santa Monica, CA 90405, (213) 399-2083.

The member authors whose articles appeared in August and September issues of *Age* intended to have their material appear in the official Society publication. We assure them that in the future no submitted material will appear in other than *SCCS Interface.*

Very Truly Yours,
The Board of Directors
SCCS Interface
October 18, 1876

A FIXIT "KIT" FOR MARK-8 DOCUMENTATION

Dear Jim, Sept. 17, 1976

I have been reading with great interest the issues of *DDJ*! I have an offering for "BUGS & FIXES." I have put together a modifications/corrections kit for the MARK-8 to fix up the over 50 typos in the schematics, errors in design, and errors in instructions. It includes instructions (11 pgs), complete set of new schematics, and a parts kit. The cost is set to only recover costs. MARK-8s have suffered in the software marketplace due to lack of enthusiasm, which I feel is in part due to the difficulties in getting them up. This package should help the problem clear up and create more spirit (since I still want BASIC for my 8008!).

MARK-8 Corrections/Mods Package - Fixes those glitches, interrupt structure, mem. addr. levels, LED bd., buffered CPU, clock phases, console controls, etc. Includes new complete schematics, instructions, and parts (even drill bit and wire). $10. Ronald Carlson, 14014 Panay Way Apt. 225, Marina del Rey, CA 90291.

Sincerely,
Ronald E. Carlson 14014 Panay Way, Apt. 225
Marina del Rey, CA 90291

BYTE OFFERS AN *EXCITING* PROPOSAL

Machine-Readable Programs in Magazine Format

Praise by Jim Warren, Editor, *DDJ*

OK, folks . . . are you ready to throw away those cantankerous and expensive paper-tape readers? Are you ready to give up those cat-naps you take while waiting for programs to load from your kid's audio cassette player (you *do* have a megabyte of memory, don't you?)? Then look to *Byte** for a better way!

The November issue of *Byte* magazine carries an article by Walter Banks and Roger Sanderson of the University of Waterloo, and Carl Helmers of *Byte,* proposing an idea that should cause the hobbyist to gleefully reposition their prayer rugs in the direction of 70 Main Street in Peterborough, New Hampshire: a super-neat method for publishing machine-readable information. Walter and Carl are proposing that the bar-code scanning techniques already in widespread use in automated grocery checkout systems are equally applicable to *publication* of machine-readable programs and data.

The basic idea is that programs and data that are of widespread interest can be encoded in a standard bar-code format, printed in a book or magazine (presumably with the human readable form on nearby pages), and loaded into an individual's home computer by simply waving an optical scanning wand over the machine-readable pages. Programs and data could then be truly "published" – printing them instead of using the far more expensive and less convenient punched or recorded formats. The reading mechanism – the scanning wand – has the advantage of no mechanical parts, depending on the human hand for its motive power. It's simple; it's nonmechanical; it should be cheap. Data transfer rates are obviously limited only by the speed of the hand and the speed of the processor that is interpreting the input from the scanner.

This is not a future fantasy. The technology is already well-developed, both for printing of machine-readable information and for inexpensive optical scanners. Optical scanning of printed information has been in use for some years in the banking industry. There, total reliability is an absolute requirement, and the encoding format and scanner design they use is considerably more complex than is necessary with bar-codes and bar-code scanners. Bar-code techniques have proven sufficiently reliable that they are in wide-spread use in grocery checkout facilities, where accuracy is a must (demanded by the paranoid consumer was well as by the food retailers), and where sloppy usage must be assumed.

This is not an idle proposal or one-shot in the dark by *Byte.* The November article is an explicit, detailed, nuts-and-bolts article. The December issue of *Byte* will include samples of machine-readable code in several experimental formats, an article on signal processing for optical scanning of bar-codes, and the specifics of the software that is necessary for reading bar-coded information. Articles in the immediate future are sure to include complete details for the construction of bar-code scanning wands and their interfaces.

We cannot praise this proposal too highly. If this technique, in fact, proves feasible and reliable, it will provide a *significant* breakthrough for the problem of distribution of machine-readable software *and data.* Note: The import of this for the forseeable future well may be in its facility for distribution of *data,* rather than programs – e.g., census data, voting records, mathematical and engineering tables, encyclopedias, want ads, library indices, case law citations, you name it – all types of reference materials that it would be desirable to be able to search and access via machine.

COMPUTER CONTROL OF TAPES HAS MUCH WIDER USE THAN MERELY FOR MUSIC SYSTEMS

Dear Jim, Sept. 20, 1976

I read your excellent article: "Computer Control of Music Tapes for Your Home Stereo" in *DDJ* Number 8. I think it is really a realizable fantastic fantasy. However, I have some objection to the title (and the emphasis) of the article.

Had you titled it: "Computer Control of Bach's Music Tape for Your Home Stereo on the Second Floor," I would have objected even more. The hardware you described is a computer controlled tape deck that can handle both digital and analog recordings. The software you proposed is also a very general file system. As you have mentioned in your "bells and whistles," this system would be ideal for many types of computer-aided instruction. I would also use it to play computer games, and many many more. Wouldn't it be great to hear Dr. Spock talking when you play Startrek? I would also use the same system to save all the programs and all my secret files. In this case the analog part may be of no great value, but my stereo on the second floor just might announce: "We are now loading a version of TINY PASCAL dated April 1, 1977." Anyway, my point is, don't limit such a great system to "Music Tapes" or to "Home Stereo."

Another nitty gritty: On Page 5, you seem to imply that the $199 and the $299 packages from Triple I also include two tape transports. From what I know, only the $189 package includes two transports. You get only one transport in the deluxe model packages.

Sincerely yours,
Linchen Wang

I debated phrasing the article in this more general applications context, but decided to keep the main article "narrow-minded" and merely point out the much more general applicability of the system I outlined. I did so because I didn't have the time or space to discuss the wider applications in the detail that I felt would be necessary to a more generalized article.

Yer right on how many transports are included in each package. The $189 package includes two fixed-speed transports, but the $199 and $299 packages include only one transport. What's worse, the prices have gone up . . . but they're still a good deal (see Phi-Deck article elsewhere in this issue).

WE HAVE SPEECH SYNTHESIS . . . SOON: SPEECH INPUT

We hear . . . straight from the quadraped's mouth . . . that a speech recognition experimentation system will be placed on the market early next spring. In kit form, it will cost well over $500 and will plug into the S100 bus.

PUBLICATION DETAILS DESIGN OF A CONTROL PROCESSOR FOR A MICRO-COMPUTER NETWORK

The Computer Systems Synthesis Group out of UCLA's Computer Science Department has recently released a 231 page tech report by R. Fenchel entitled, "A System Control Processor for a Microcomputer Network." It discusses the design of a control processor for such a network, to be used as an education tool in a computer science lab. You can probably obtain a copy without cost (while they last) by writing the CS Department in the School of Engineering & Applied Science, UCLA, Los Angeles, CA 90024.

*Are there any of our readers who *don't* know about *Byte* magazine, 70 Main St., Peterborough, NH 03458, $12/year?

Why Hassle With Those Single-Purpose Cassette Interfaces?

USE AN ACOUSTIC COUPLER TO READ/WRITE TAPE CASSETTES

Jim Warren, Editor

Steve Moore* just phoned in a hot idea. Why not use a data communications modem or acoustic coupler to read from and write to audio cassettes?

Here are the advantages: By doing so, suddenly all of the "recording standards" problems disappear. The standards for couplers and modems have been accepted and in use for some years – and are well debugged. Why waste our time haggling over which homegrown standard to adopt, when we can "steal" the standards that have been proven in industrial use for well over a decade?

Couplers and modems are specifically designed to interface to a byte-oriented digital device. Plenty of them are around that are already built to plug into a 20 mA current loop or RS-232 standard interface. It should be a simple matter to modify the master/slave circuitry (see the "gotchas", next section) so they can talk to a computer instead of a terminal. (Quick! – all you hardware fanatics: send in the hardware details to guide us naive systems fanatics in making the necessary changes).

Modems and couplers have been around for so long that a number of them are on the used equipment market. Some months ago, Walt Gruninger at the Minicomputer Exchange (154 San Lazaro Ave., Sunnyvale, CA 94086, 408/733-4400) told me that couplers could easily be had for about $100.

It's a quick way to gain hardcopy facilities when you have no hardcopy device. Here's how: have your system dump a text file into your kid's $19.95 audio cassette via a coupler or modem. Take the whole thing over to anybody's coupler-equipped time-sharing terminal. Play the tape into the coupler (via a telephone handset that you scrounged from a surplus phone), and watch the pretty hardcopy be printed. The cassette tape is just acting as a hand-carried "telecommunications system."

Once the coupler or modem is interfaced to your coupler, of course it can easily be used for telecommuning with another computer or a central program and data storage facility. Such central repositories are already being discussed as (1) a good solution to the problem of home computers having access to continually updated programs and data, and (2) an appropriate project for any of the larger clubs (if "hams" can get together in constructing co-op relay stations, why can't we cooperate in building machine-accessable central repositories?)

A quick check with an old analog engineer friend, down in Silicon Gulch, assured me that using this technique to handle data-rates up to 300 baud would present no problems, even when using el-cheapo cassette units and audio tapes. Note that this is the same (rather slow) data-rate as the *"Byte* standard." It is obviously no problem since, after all, couplers are rated up to 300 baud and are explicitly designed to function reliably over scuzzy, unconditioned, lowest-bandwidth telephone lines. Modems are currently available that will run up to 9600 baud over conditioned phone lines. My analog friend hedged somewhat on whether or not such higher data-rates would present problems on audio tapes. Again, I call on you hardware types for the necessary details to make this fantasy a reality.

And now, the hidden gotchas: First of all, the garden variety acoustic coupler is built with the electronic protocols for its analog end to slave to a master computer over the telephone handset, and its digital end to speak in full-duplex or half-duplex to a terminal. Its protocol circuitry must be modified so that its digital port will be the slave to the computer and its analog part connect to the "terminal cassette." Alternatively, one might purchase a "master modem" that is normally connected to a time-sharing computer; however, these are considerably more expensive, probably have unneeded bells and whistles, and are less available on the used market.

If trouble appears in the analog end of this system, it will be considerably more difficult for the novice to debug and fix than is the case with strictly digital circuitry, or with the *Byte* or Tarbell cassette standards. If you use an acoustic coupler, you must homebrew a connection between it and the record and playback "I/O" of your cassette. This may require some amplification circuitry.

Now it's up to you. It is an interesting and valuable project that is obviously well within the limits of current technology and a hobbyist's budget – a realizable fantasy. When you get it up and running, why not share your implementation with everyone via an article in *DDJ?* Incidentally, the quicker a computer-coupler interface becomes widely available for home computers, the quicker we will see the creation of the machine-accessable program and data repositories that I mentioned earlier – yet another "realizable fantasy."

*Steve Moore is a consultant with Moore Research, P.O. Box 1562, Sacramento, CA 95814, (916) 441-1890.

U.S. POSTAL SERVICE

STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION

(Required by 39 U.S.C. 3685)

1. TITLE OF PUBLICATION: Dr. Dobb's Journal of Computer Calisthenics & Orthodontia	2. DATE OF FILING: 76Nov8
3. FREQUENCY OF ISSUE: monthly except for July & December	A. NO. OF ISSUES PUBLISHED ANNUALLY: 10 / B. ANNUAL SUBSCRIPTION PRICE: $10
4. LOCATION OF KNOWN OFFICE OF PUBLICATION *(Street, City, County, State and ZIP Code) (Not printers)*: 1010 Doyle #9, Menlo Park CA 94025	
5. LOCATION OF THE HEADQUARTERS OR GENERAL BUSINESS OFFICES OF THE PUBLISHERS *(Not printers)*: same as No. 4	

6. NAMES AND COMPLETE ADDRESSES OF PUBLISHER, EDITOR, AND MANAGING EDITOR

PUBLISHER *(Name and Address)*: People's Computer Co., 1010 Doyle #9, Menlo Park CA 94025	
EDITOR *(Name and Address)*: Jim Warren,	"
MANAGING EDITOR *(Name and Address)*: Jim Warren	"

7. OWNER *(If owned by a corporation, its name and address must be stated and also immediately thereunder the names and addresses of stockholders owning or holding 1 percent or more of total amount of stock. If not owned by a corporation, the names and addresses of the individual owners must be given. If owned by a partnership or other unincorporated firm, its name and address, as well as that of each individual must be given.)*

NAME	ADDRESS
People's Computer Company	

8. KNOWN BONDHOLDERS, MORTGAGEES, AND OTHER SECURITY HOLDERS OWNING OR HOLDING 1 PERCENT OR MORE OF TOTAL AMOUNT OF BONDS, MORTGAGES OR OTHER SECURITIES *(If there are none, so state)*

NAME	ADDRESS
na	

9. FOR COMPLETION BY NONPROFIT ORGANIZATIONS AUTHORIZED TO MAIL AT SPECIAL RATES *(Section 132.122, PSM)*

The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes *(Check one)*

☒ HAVE NOT CHANGED DURING PRECEDING 12 MONTHS ☐ HAVE CHANGED DURING PRECEDING 12 MONTHS *(If changed, publisher must submit explanation of change with this statement.)*

10. EXTENT AND NATURE OF CIRCULATION	AVERAGE NO. COPIES EACH ISSUE DURING PRECEDING 12 MONTHS	ACTUAL NO. COPIES OF SINGLE ISSUE PUBLISHED NEAREST TO FILING DATE
A. TOTAL NO. COPIES PRINTED *(Net Press Run)*	3125	5000
B. PAID CIRCULATION 1. SALES THROUGH DEALERS AND CARRIERS, STREET VENDORS AND COUNTER SALES	810	1426
2. MAIL SUBSCRIPTIONS	1332	2670
C. TOTAL PAID CIRCULATION *(Sum of 10B1 and 10B2)*	2142	4096
D. FREE DISTRIBUTION BY MAIL, CARRIER OR OTHER MEANS SAMPLES, COMPLIMENTARY, AND OTHER FREE COPIES	50	60
E. TOTAL DISTRIBUTION *(Sum of C and D)*	2192	4156
F. COPIES NOT DISTRIBUTED 1. OFFICE USE, LEFT OVER, UNACCOUNTED, SPOILED AFTER PRINTING	933	844
2. RETURNS FROM NEWS AGENTS	0	0
G. TOTAL *(Sum of F, F1 and 2—should equal net press run shown in A)*	3125	5000

11. I certify that the statements made by me above are correct and complete. SIGNATURE AND TITLE OF EDITOR, PUBLISHER, BUSINESS MANAGER, OR OWNER

ITALICS IN VIDEO DISPLAYS

One possible enhancement of character generation in TV typewriters is the incorporation of an italic mode. The same ROM could be used to produce the basic dot patterns for both italics and non-italics, only the character timing would change. Figures 1 and 2 show how text strings would look in both modes. In the italic mode, successive lines of each character would be displayed with a different time delay. Assuming a 7 by 9 dot matrix, the first line of each character (i.e., the top line) would be displaced by 4 dots to the right. Line 2 would be displaced by 3.5 dots (i.e., three and a half dot-clock cycles), and so on. Line 9 would have no displacement. A shift register IC could be used to implement the displacement, and an embedded control character (such as CTRL I) could be decoded to turn the mode on or off. The regular and italic modes could both be used in the same line of text with appropriate control of transitional timing, although this would complicate the logic required.

by Jim Day

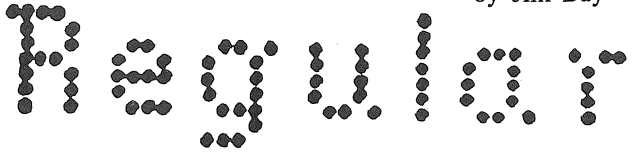


Figure 1. Regular Mode

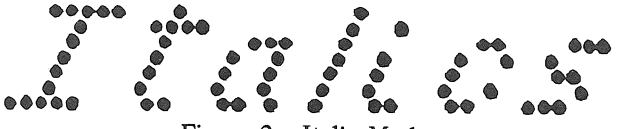


Figure 2. Italic Mode

SCROLLING MOD FOR TVT-2's

Dear Sirs, Oct. 7, 1976

The TVT-2 is the most popular video terminal used by computer hobbyists today. Until now, the users have had to settle for the 'page concept' with their terminals. Your readers might be interested in the fact that now they can add scrolling to their TVT-2. A fully assembled scrolling modification board (model SM-2) is available from Lenwood Computer Systems, P.O. Box 67, Hiawatha, IA 52233. A complete set of instructions is supplied. The cost of the SM-2 is $20.00 plus $1.50 for postage and handling.

Thank you for your time.

Jay G. Francis, P.O. Box 67
Lemwood Computer Systems, Hiawatha, IA 52233

64 X 32 VIDEO DISPLAY KIT

Gentlemen:

I thought that some of the *DDJ* readers interested in video displays might want to look into a kit sold locally here in Dallas. It is a 2K x 8 bit parallel I/O (32 lines of 64 chars); it may be optioned for RS232 also.

The main reason for going to this unit was because of the several control codes that allow blink by code and blank. The blank/unblank allowed me to not only not use up my own rather limited core (RAM, actually), but to use the screen for extra RAM, as the unit will operate at machine speed – I'm using the INTEL 8080 prototyping kit.

Readers interested can write to the company at this address:

IOR
P.O. Box 28823
Dallas, TX 75228

Sincerely,
D. Moore
Dallas, Texas

64-CHAR MOD FOR TVT-II'S, NOW & SCROLLING, SOON

Gentlemen:

We would like to thank you and your readers for the interest in the TVT-11 64 character modification article that you published in your No. 6 issue. The response has been tremendous. After experiencing some initial problems with typo errors and delayed shipments we are now meeting with your 3 week delivery schedule. On November 1 we found it necessary to increase the price of our boards to $6.50 for the 64 character board and $13.50 for the 2K memory boards. Printing costs have also required us to charge $2.00 for the instructions if requested without ordering the boards **[a corrected copy of the necessary instructions is now available]**.

We have received many requests for a scrolling modification for the TVT-II and we are happy to announce that we will have one ready to ship out before the end of the year. The board is set up such that only five jumpers are required to install it. This was accomplished by having the board plug into sockets which replace IC's 34 and 41 on the mainframe. These IC's are relocated on the mod board. The board gives bottom line scrolling with the new line coming up clean. Normal or scrolling modes are switch selectable with the scrolling not beginning until the page is full. Line feed is disabled when in the scrolling mode. It appears that the board will retail at $20.00 with sockets and instructions although this is not yet firm.

Thank you,
David O. Valliere, Box 4241
Digital Designs, Victoria, TX 77901

VTT GROUP BUY

Dear Jim, Oct. 27, 1976

In response to our conversation on the phone today, here are all the details on the VT4000 group buy. There are two different buys available at this time. One is the VT4000B, a complete crt terminal with a Motorola 12" monitor, completely assembled, tested and ready to use. The second is for the do-it-yourself types. It consists of the five PC boards without parts, the power supply and the cabinet for the VT 4000A. To qualify for the group buy price, we will have to buy at least 10 of the buy or buys we choose. To try and clarify further, we cannot combine the two to get a total of 10. If the minimum of 10 is not reached by 15 January 1977, all money will be returned. To qualify as an order, full payment must be made at time of order.

Prices:

VT4000BA Assembled Terminal regular retail	999.95
less 20%	199.95
	800.00 + 6.5% state tax + 2%
TOTAL	868.00
Bare bones lit separate parts regular retail	240.00
less 10%	24.00
	216.00 + 6.5% state tax + 2% handling fee
TOTAL	234.36

The above prices include delivery in the Bay Area, outside the Bay Area will be sent freight collect. If shipped outside Bay Area subtract ½ of the handling charge.

Make checks payable to: Video Terminal Technology. Mark on lower left of check: Homebrew Computer Club Buy. Send orders to:

Norman Walters
3107 Laneview Dr.
San Jose, CA 95132

NEW COSMAC COMPUTER

At last, someone has come out with a microcomputer based on the RCA 1802 (COSMAC) and suitable for many hobbyists. Produced by Infinite, Inc. (Box 906, 151 Center St., Cape Canaveral, FL 32920), this machine has a 4-digit hex readout and integral hex keyboard. Standard on-board memory comprises 256 bytes of RAM, externally expandable to 64K. Additional features include DMA as well as serial and parallel I/O. Assembled and tested, the price is $395 with case and power supply, $249.95 for a complete kit, or $179 for just the MPU board.

MILITIA MAY AID HOBBYISTS COMPATIBILITY PROBLEMS

Dear Jim, Sept. 7, 1976

You are probably aware of the WESCON Session II paper described in the attached extract from Electronic Design (below). Looks like the military may be giving us a hand with standardization.

I enjoyed meeting you at Personal Computing '76.

Best wishes,

Joe Gilbreth 1229 Vista Lane
Birmingham, AL 35216

Recommendations for use of a common bus system will be made in Session 11, but in another context—for the standardization of military microprocessor systems. That will be proposed in a Session 11 paper, "Compatibility Among Families of µPs", by Hank Malloy, military program manager, Intel Corp. Malloy is also chairman of a newly organized task force on military microcomputer LSI, which is sponsored by the Electronic Industries Associates and the National Electronic Manufacturers Association.

To achieve any kind of standardization it is essential that bus structure characteristics be specified, Molloy will argue. Also, high-order languages will have to be used.

An example of how such languages can contribute to standardization, Molloy will point to PL/M. Two popular 8-bit µPs are the Intel 8080 and Motorola's 6800. While PL/M was generated by Intel for the 8080, PL/M compilers are available to translate the syntax into object code for the 6800. [And Signetics 2650.]

The EIA/NEMA task force will study drafts of two new MIL-M-3851 microprocessor detail specs, the /400 for Motorola's 6800 and the /420 for Intel's 8080.

MORE COMMENTS ON PROC.TOLOGY SOFTWARE, PLUS SOME NOTES ON CASSETTE TAPE QUALITY

Dear Jim,

DDJ has become the best newsletter for the computer hobbyist. None of the commercial magazines can approach the wealth of information you provide. I enjoy every issue, especially the letters. Keep up the good work. [Aww, geee fellas . . .]

Some more comments on Processor Technology software. I think the PT people have done a great job providing reasonably priced software. It seems that their programs are not thoroughly debugged. The source listings for FOCOL and BASIC do agree with the paper tapes.

The problem with the PT BASIC INT function mentioned in *DDJ* No. 8 can be corrected as follows:

```
AINT    LDAX    B
        SUI     129
        JP      AINT1
        XRA     A
        MVI     D,5
AINT2   STAX    B
        DCX     B
        DCR     D
        JNZ     AINT2
        STAX    B
        RET
```

Thanks for Fred Greeb for this fix.

In addition, I've come across two more bugs:

1. Formatted print will not work with fractional values.
   Example - %Z2% Variable = 1.097 Output = 1.10 OK
   Variable = .097 Output = .0010 OOPS!
2. When a variable in a FOR/NEXT loop is decremented and becomes zero it is not recognized as zero.
   Example - 10 FOR I=2 TO -2 STEP -1
   20 IF I=0 THEN . . .
   The relation on line 20 never becomes true. The zero is apparently a "negative zero" since
   20 IF ABS(I)=0 THEN . . .
   will work.

If any of the *DDJ* readers have a solution I'd like to hear from them. A letter to PT regarding these problems has not been answered.

Realistic Supertape has been recommended in some hobby magazines as suitable for digital data recording. The October 76 issue of *Consumer Reports* contains a test of audio cassettes. Recording music and digital data are not directly related. However, it is interesting to note that Supertape was rated below average in two important factors - output uniformity and freedom from dropouts. From personal experience I'd have to agree with the test findings.

The four top rated cassettes were BASF Studio Series, Maxell UD-XLC60, Scotch Master, and TDK Super Avilyn SAC60. I'm not sure this can be printed in *DDJ* since it is copyrighted info.

Happy computing.

Adolph P. Stumpf 5639-A Ute
Glendale, AZ 85307

[My impression is that one may copyright text, not information. — Jim]

FRIDEN DOCUMENTATION FOR $10

Dear Editor, Oct. 31, 1976

Some months ago I purchased a Burroughs-Friden Printer-Keyboard and the associated interface electronics on the surplus market. The model number is 9530-2. The cost was in the vicinity of $300, and looked like a pretty good deal for a hard copy unit. The major shortcoming is the lack of any documentation. For effective use of the unit with home computers some changes are necessary, but are virtually impossible to accomplish without adequate documentation. I spent nearly all of my spare time for the last half year on the incredible task of deciphering the circuits on the interface boards. There are over 300 integrated circuits (obsolete types) on the boards. It was the hardest puzzle that I ever worked on.

It is likely that there are other computer freaks who have bought similar units and are in need of documentation. For $10, I will send a copy of my documentation to anyone for his or her personal use. The documentation includes comments on almost all of the inter-board wires and logic diagrams of the boards and typewriter switches. It does not include explicit instructions for modification of the boards for home use, but perhaps I can generate that later.

Sincerely,

Robert L. Smith 2300 St. Francis St.
Palo Alto, CA 94303

Does anyone have *original manufacturer's* documentation for these units? — Editor

$3000 FOR 2,400 LINE PER MINUTE PRINTER

Houston Instruments has 80-column and 132-column printers that print up to 2,400 lpm and up to 1,400 lpm. They say their interfacing is explicity designed for easy connection to micros.

Houston Instruments is located at 1 Houston Sq., Austin, TX 78753.

THINKING OF OPENING A COMPUTER STORE?

Before you do, consider the following financial figures. These were generated in September, 1976, by an independent team of professional cost analysis consultants. They are *projected* or *"reasonable expectation"* figures for two classes of computer stores; a $20K/month store and a $30K/month store (gross). They are based on a number of in-person and in-depth telephone interviews with a large number of existing computer stores.

Average Earnings For An Average *Month*

Gross Sales	$20,000.00		$30,000.00
Cost of Goods Sold	13,600.00		20,400.00
Gross Profit	6,400.00		9,600.00
Deductions:			
Refunds, Bad Check	$ 15.00		$ 25.00
Sales Expenses:			
Personnel Advertising	10.00		15.00
Salaried Employees (2)	1500.00		1500.00
Bonuses	150.00		250.00
Royalties	1000.00		1500.00
Subtotal	/$2675.00/		/$3240.00/
Operating Expenses:			
Advertising (2%)	400.00		600.00
Automotive	35.00		50.00
Dues & Subscriptions	7.50		7.50
Entertainment	10.50		25.00
Equip. Rental	30.00		40.00
Insurance	75.00		75.00
Interest	7.50		10.00
Office Supplies	10.00		20.00
Postage	17.50		25.00
Printing	20.00		30.00
Prof. Service (Ace/Lease)	75.00		75.00
Rent	450.00		600.00
Taxes	35.00		50.00
Telephone	100.00		150.00
Travel Expense	50.00		75.00
Utilities	50.00		75.00
Subtotal	/1373.00/		/1907.50/
Net Profit (11.7%) Monthly	2352.00	(13.8%)	3462.50
Yearly	$28,224.00		$41,550.00

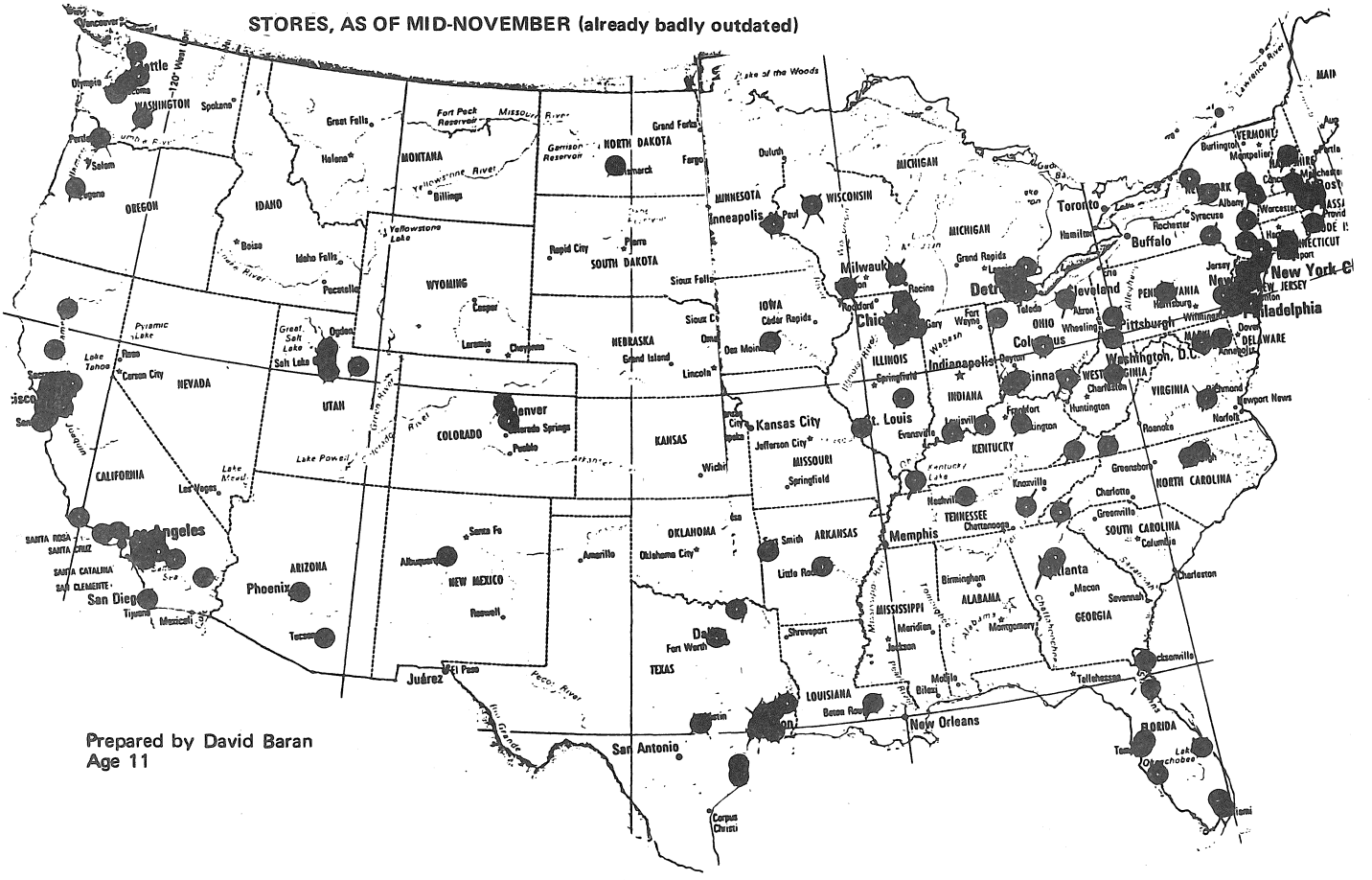
NEW CANADIAN COMPUTER STORE

The Computer Shop (of Calgary) is a brand new store serving the Canadian Rockies and western plains area. They carry a number of product lines, and hope to offer some of their own Canadian-made products in the near future. Austin L. Hook, The Computer Shop, 3515–18th St., SW Calgary, Alta., T2T 4T9, Canada, (403) 243-0301.

"PERSONAL" COMPUTERS ARE SHOWING UP IN SCHOOLS

The San Jose Unified School District is busying 14 Western Data Handlers, assembled. It already has ten IMSAI's and a Polymorphic. It has originally been considering expanding a PDP-8 into a TSS-8 system, but decided to purchase these 25 computers, instead – for the price of that TSS-8 expansion.

Furthermore, Bob Albrecht noted, "SMRT won't hurt San Jose." (SMRT is the Single Message Rate Tariff that Pacific Telephone is about to inflict on users of business telephones who make lengthy calls. See August, 1976, *DDJ*.)



JIM McCORD REPORTS ON THE LSI-11

Dear Bob and Jim, Oct. 7, 1976

To follow up my conversation with Bob of a couple of weeks ago, this is to tell you about the LSI-11 stuff.

At last count there were about 15 people in the S. Calif. area who were using the LSI-11. I understand that there are about an equal number in the Bay Area. Other than those two groups I know of no other "large" bodies of hobbyists using the machine, although there are undoubtably isolated people around the country who bought them from various distributors. Perhaps an announcement in *PCC* or *Dr. Dobb's* will help pull us together.

The S. Calif. group bought their machines from a company called Applied Information Development, a subsidiary of SDC. AID is apparently building something that incorporates the LSI-11 and is selling the components partly as a way to get their own unit costs down. We got a 25% discount of quantity one price with a $5K order, and some smaller orders have since been filled at the same discount. Whether they would still do this for other groups I do not know, but probably they would. (Amateurs pay *cash.*) I have also seen other distributors advertising "club discounts" on the LSI-11. By the way, we went this route after trying for almost a year to put together a group of 50 people to buy directly from DEC and never succeeding.

There is a common belief that the LSI-11 is too expensive for hobbyists. I don't agree. For about $1K, you can get a processor, 8K bytes of memory and a serial I/O card, and a backplane, fully assembled to industrial standards, that works when you plug it in. It took me 15 minutes to go from box to teletype. The machine has a very nice monitor, and for an extra $100 or so you get hardwired fixed and floating point instructions, for those who are into that. *Plus,* you get the very elegant and powerful instruction set of the PDP-11, all of the system software that has been developed for the 11 (at a price, of course), and the DECUS library which is full of 11 software and is going to get a lot fuller. All in all, I think it's a pretty good deal.

There are some disadvantages, of course. If the machine breaks you probably have to ship it back to DEC for repair. (DEC claims a very long MTBF, but who knows?). I don't know what a nominal repair charge would be. Until somebody builds a LSI-11 – to – Altair bus interface, we won't be able to use all the neat hobby peripheral cards. Memory is somewhat more expensive than for hobby machines, although it too comes fully assembled and checked out from a variety of vendors. You have to supply your own power supply and box, although that isn't a big deal. Also, some of the most desirable software like the BASIC interpreter is still pretty expensive. I think DEC should consider releasing the papertape stuff to DECUS – probably they have recovered the cost by now! In all, though, I think that the LSI-11 has a lot to recommend it to the hobbyist, particularly to those who are more into programming than hardware.

So far no really creative applications for the machine have emerged from our group, since most of us are still working on developing auxiliary hardware like terminals and stuff. Peripherals include a few TVT's and TTY's, papertape readers and punches, and cassettes. Three of us have built the InteColor 8001 intelligent color terminal kit and are using it as our main I/O device (that's a story for another day). The peripheral that most of us would like to get is, of course, a floppy disk, but so far we haven't found anybody who makes an affordable controller for the LSI-11. That shouldn't be too far off, though.

Anyone who has an LSI-11 or is interested in one is welcome to write to me. So far there is no organized newsletter for the machine but undoubtedly one will emerge when enough people are interested. DEC will create a DECUS Special Interest Group (SIG) for hobby users of the 11 or 8 or both, which would take care of nuisances like printing and mailing, but we need a few more users before that becomes reasonable.

See ya,

Jim McCord 3710 State St.
SysteMetrics, Inc. Santa Barbara, CA 93105

P.S.

Jim, thanks for the stuff on PerSci. They had an ad in *Interface* this month, offering drive and controller for just over a kilobuck. It's a really fantastic intelligent controller, requiring practically no support software in the host machine. If I can't find a compatible controller for my 11, I may go this route, writing my own drivers.

POSTSCRIPT TO 'COPYRIGHT MANIA'

Dear Jim, Aug. 25, 1976

I am writing this letter as a postscript to the article 'Copyright Mania' in the May issue of *Dr. Dobb's.* I became rather attracted to TRAC (trademark of Rockford Research, Inc., and don't you forget it) and therefore wrote to Mooers asking for information regarding the development of a TRAC processor. Two months later, I received a copy of a 'License Agreement for permission to use Rockford Research copyrighted writings on TRAC language in academic experimentation.' What it consisted of was an agreement that Rockford Research would not sue me if I signed the agreement *and* sent them $10. That (the promise that they won't sue me) is all I get for my $10 (manuals are another $15). Also, once I finished the TRAC processor I could not: "publish, reproduce, resell, lease, give, lend, circulate, or license the . . . [TRAC processor] . . . or any portion thereof in any manner or on any medium, which shall include but not be limited to copies, tapes, films, computer program library deposits . . ." (there was about a paragraph more). Anyway, that rules out sending to *Dr. Dobb's,* which was my idea for the processor from the start. The agreement also wanted me to agree not to challenge the Rockford copyright.

In short, I have no objection with a person or group copyrighting a program, but this seems a bit excessive.

Thanks for the time, and keep up the good work.

Yours,

Chris Pettus PO Box 611
Malibu, CA 90265

PRAISE FOR PALO ALTO TINY BASIC & TINY TREK, AND A QUIBLET ABOUT THE VDM SOFTWARE

Dear Jim, Oct. 5, 1976

PALO ALTO TINY BASIC and TINY TREK have to be the best $4.00 investment I ever made! I'd like to recommend it to all *Dr. Dobb Dobb's* readers. (For further details, refer to *Dr. Dobb's,* May 1976, with an addition of software for a VDM display in June/July 1976.)

The tapes came back within 4 days from the Community Computer Centre (which must be a record for 'Trudeau's Turtles'), and everything worked immediately. The abbreviating possibilities of P.A.T.B. really make for conpact programming (P. instead of PRINT, for example), in conjunction with multiple statements per line.

One thing I would have liked to have seen would have been simple strings for inputting and outputting words, names, etc., but one can't have everything in less than 2K, I guess. (Any chance of Li Chen Wang re-considering . . .?)

I haven't had too much use out of TINY TREK – mainly because my kids won't let me have a turn! However, the times that I have played, I have thoroughly enjoyed it, and it ranks up there with the other versions I have played (STARTREK, and STARTREK 3D on an AMDAHL 470). As a matter of fact, it is extremely difficult to win, and that increased the enjoyment (with the frustration).

Another problem in using the VDM software given in June/July 1976 *Dr. Dobb's,* is that when listing a long program the screen goes zzzip! and all you catch is the last few instructions that remain on the screen. A delay, or a hold feature would be nice.

Still, for 6K of memory I have hours of fun – or at least my kids do. I'm reduced to playing after lights out for them. What the heck can I do with my other 10K?

Sincerely,

Basil R. Barnes, VE6BB Box 1226
Bonnyville, Alberta
CANADA, T0A 0L0

P.S. Can I obtain Mr. Wang's address? [Dr. Lichen Wang, 150 Tennyson Ave., Palo Alto, CA 94301, (415) 321-6983]

A GOOD RESPONSE TO COMPLAINTS ABOUT TARBELL TAPE UNITS

Dear Jim, Sept. 19, 1976

Thank you for giving me the opportunity to reply to the notes about my cassette interface in your Volume 1, Number 8 issue.

I believe that no product is ever perfect, so I continually revise both the documentation and the interface itself. Since I started delivering these units over a year ago, I have gone through four revisions of the boards, and at least six revisions of the manual. These changes were largely the result of complaints, suggestions, and returned survey forms, which are at the end of each manual. The first ten kits especially, were followed very closely, and the owners were asked to immediately inform me of any problems they had with either the manual or the board. In this sense, the kits *were* "tested on persons unfamiliar with the device."

Although I realize that the term is a relative one, I don't feel that the implementation of this device has been at all "sloppy." Of course, I've had my share of problems, like any of the other manufacturers, but I've made every attempt to follow up what I consider good design practices, and to make the system as clean as possible.

Unfortunately, I did have a run of boards that had bad plated-through holes, and got through my inspection undetected. I have since discontinued my relationship with the manufacturer that produced these boards, and selected another. My first revision D cassette interfaces were delivered September 3, 1976 (before *Dr. Dobb's* Number 8). The boards in these kits, one of which is enclosed, are far superior to the previous ones, and the plated-through holes look beautiful. Revision D also includes all the latest modifications, including several unused inputs connected to pull-up resistors. The connector pin alignment has also been corrected slightly.

Several months ago, I contracted with someone to completely rewrite the manual. The rough draft is now being reviewed, so it will probably be ready to print in about a month. This manual provides new information, such as siagrams for all the integrated circuits, step-by-step instructions for the beginner, and a more thorough theory of operation section. Although my present manual is not in a professional format, I am proud of the fact that it is chock-full of the kind of information a hobbyist needs to get his interface up and running and useful. The new manual will be even better, and some of the pages have already been added to the present manual.

One page of the manual starts: "If you cannot make at least ten 8K-byte transfers with no errors, you have a problem, and the items below may be of some help:" This is followed by several items to check. The last two items on this page state: "If you still have problems, please return the unit, preferably with your cassette recorder, and I will get it operating perfectly for you without charge. If you are completely dissatisfied, you may return the interface for a refund within 90 days after you accepted delivery."

I don't know if there's another manufacturer that stands behind his product like this, but I think it attests to my confidence in the Tarbell Cassette Interface. I have not charged one penny for repairs yet, and in all the units I've shipped, only one has asked for (and received) a refund. I sincerely believe that most of the people with these units are completely happy with them.

I completely support your suggestion to write or phone me directly. *Please*—if you have problems with your interface, and we can't seem to get it going over the phone or by mail, send it to me for repair. There have been some units that have been difficult enough to repair that I've actually supplied a replacement unit, at no cost. I don't really see how you can lose when you buy one of my interfaces.

Sincerely,

Donald E. Tarbell
(213) 832-0182
Tarbell Electronics
144 Miraleste Dr., No. 106
Miraleste, CA 90732

CONFERENCE ON COMPUTERS IN HUMANITIES

Papers and participation are being sought for the Third International Conference on Computers in the Humanities to be held Aug. 2-5, 1977, at the University of Waterloo, Waterloo, Ontario. Send papers or abstracts to Prof. Paul Bratley, Dept. D'Informatique, Universite de Montreal, Montreal, Quebec H3T 1J4 by January 15, 1977.

TARBELL PRAISE, A FANCY DISASSEMBLER, & AN APL CHARACTER QUERY

Dear *DDJ*, Sept. 26, 1976

Just a quick note on the Tarbell interface about which you say you have been receiving a lot of complaints: mine worked the first time I tried it, and refused to drop so much as 1 bit when fed by a tape recorder with a variable speed control. I could go 20% slow, and about 10% fast, with no trouble. Only by intentionally trying, could I get it to drop a bit to see if my checksum routine was working. In daily use for about 2 months, it has dropped a bit only once. It is a fantastic peripheral. I use it to back up floppy disks as it is the only device that is 1) fast enough; 2) cheap enough; 3) reliable enough. I had no 'non-plated-thru hole' problems. A friend says Tarbell left some TTL inputs floating, which causes noise susceptibility, but I have not had this problem.

Keep up the fantastic work on supplying the hobbyist community with public domain software, and P.S. are you interested in a disassembler which I wrote and commented? It is 8080 based, uses sense switches to determine when to generate instructions, when to generate DB's with ASCII, or DB's with hex. It can be used as a one-pass process to just see object, or can be used as 3 passes: 1) every address reference (JMP, LXI, CCLL, etc.) is placed in a symbol table compatible with Processor Tech Package No. 1; 2) a pass to edit the symbol table and change default labels (Lxxxx) into meaningful ones if you have some knowledge of the source code—this pass is entirely optional; 3) do the actual disassembly, with most labels put in, and all LXI's, JMP's etc. referencing labels. The output is a source listing, and optionally, using a sense switch, writes the source in a format compatible with Processor Technology Package No. 1.

I would appreciate you publishing a note asking if anyone knows a source for APL character generators which could be retro-fitted to a VDM.

Sincerely,

Ward Christensen
688 E. 154th St.
Dolton, IL 60419

Yes, Yes, Yes! Send us your super disassembler—including, of course, user documentation, at least nominal internal documentation, and annotated source code.

What sort of disc system are you using and how do you like it? (And, may we publish your reply?)

What sort of printer and printer software are you using? [The original of this letter had an unusual type face, and was left & right justified.]

—Jim

TARBELL TOUTED

About the Tarbell interface: I have two of them (since I have two machines) and they both work great. I recommend them.

However, neither of them worked right off. One was an early type and needed fixes; the other had a bad board and needed fixing. But Tarbell gets them right back (a couple of weeks) and they're great!

In fact, most of this stuff doesn't work immediately. None of my stuff has worked right off. That's why you should buy from someone who will back his merchandise. I've bought some used equipment and have regretted it. When you buy, buy quality and mentally add 25% for repairs unless you know at lot about this stuff that I don't know.

Say, what's your experience with molex pins for IC chips? I've not used them but I hear they work okay. Sure sounds better than buying sockets at 50 cents each.

Jim Leek
2801 F
Bakersfield, CA 93301

TARBELL TRICKY

Dear Jim, Oct. 8, 1976

A few weeks back you asked for user comments on the Tarbell Cassette Interface. Here are mine, based on a not-yet-up-and-running board.

When I first put the thing together, I had trouble getting the sync light to come on at all. So I sent the board back to Don Tarbell, asking for help. He corrected a few errors, made some modifications to the circuit, and sent it back to me — no charge. I still had some trouble getting the unit to read in data, even after setting it up with the aid of the sync light, and set it aside until I could get hold of a scope. The scope showed that adjustment was even more critical than the instructions would indicate. I was getting a good sync light reading over a wide range of settings, but the waveform was stable for only a very small range. That problem corrected, I could read in data, but still had substantial numbers of errors. I've pretty well stopped at that point, since business is taking me out of town too much to concentrate on a solution.

My observations:

— The interface is sensitive and error-prone. I assume this is the price one pays for the high speed.

— A scope is nearly essential to correct problems. This is true for all computer applications, actually, but this was the first of six boards I've assembled that required more than a little initial prodding.

— Don *is* good to his customers. I have no qualms about calling him if I can't get the error problem corrected, but want to put in my own best efforts before bothering him further.

— An article in *DDJ* mentioned that the user has to figure out that a start byte is essential. True. This can be a real problem if you don't use Don's programs, because that's the only place mention is made of it.

In short, I have mixed emotions. I appreciate the potential speed of the interface and Don's integrity in backing his product. But I would have been much farther along in getting a system running with a slower, but more fool-proof (literally) unit. Right now, I'm still using console switches, since I can't use the keyboard effectively without reloadable software.

Sincerely,
Jim Wilson San Diego, CA

Dear Jim, Oct. 14, 1976

A P.S. to a letter I wrote a few days ago about my experiences with the Tarbell cassette interface unit:

It is now up and running, apparently reliably. My solution finally was to hook it up to my hi-fi tape deck through an old Lafayette stereo amplifier that was going unused. I then used an oscilloscope to remove as much of the distortion as possible by adjusting the bass and treble controls.

This is obviously a pretty unwieldy solution. So when I finish some more important things, I plan to buy a cheap audio amplifier with tone controls (something less than 10 bucks), and use it for a more permanent installation.

Sincerely,
Jim Wilson
San Diego

Seymour Cray, designer of giant machines for CDC, and designer of the monster CRAY-I, is said to often refer to his machine as a "back-end processor."

A CLUB SURVEY: TOO SMALL FOR STATISTICAL SIGNIFICANCE, BUT WE LIKED IT

Dear Mr. Warren, October 3, 1976

Thought you might be interested in the enclosed results of our first membership survey.

Sincerely,
Robert R. Wier
Survey Chairman

Texas A&M University Microcomputer Club, Box M-9
Aggieland Station, TX 77844

These are the results of the first periodical Texas A&M Microcomputer Club Consumer Survey and Opinion Poll taken in Sept., 1976.

HOW TO READ IT: Companies are listed in each division in descending order of rating. Entries are of the form NAME (rating/no. of respondees). The ratings are determined from a number of ratings based on response speed, quality of product, pricing, complaint satisfaction, and overall quality. Companies which received less than 2 ratings are not listed. Note that it could be "safer" to deal with a company which has a rating slightly lower than some others if a large number of persons found it favorable.

In all categories, most favorable rating was 5, least favorable was 1.

COMPONENTS:
DIGI-KEY (4.58/4)
TRI-TEK (4.5/2)
JAMES (4.42/10)
S. D. SALES (4.42/6)
NEWARK (4.36/3)
ALTAJ (4.04/5)
FORMULA INT'L (4.00/2)
GODBOUT (3.9/4)
MESHNA (3.82/6)
DELTA (3.54/4)
BABYLON (3.36/3)
SOLID STATE SALES (2.89/5)
POLYPAKS (2.62/8)
OLSON (2.35/5)
B-A (2.28/5)
RADIO SHACK (1.92/11)

PUBLICATIONS:
Dr. Dobb's (4.38/4)
Byte (4.23/10)
73 (4.18/3)
Ham Radio (4.16/4)
People's Computer Company (4.10/4)
Computer Hobbyist (3.75/2)
Popular Electronics (3.65/10)
Interface (3.08/3)
Radio Electronics (2.92/9)
Creative Computing (2.88/3)

KITS AND MAJOR COMPONENTS:
SWTPCO (4.16/8)
HEATHKIT (4.16/9)
PROCESSOR TECH (3.90/2)
IBM (3.75/2)
MOS TECHNOLOGY (3.74/3)
IMS (3.73/4)
MITS (2.82/5)

"Should software be included in the price of the hardware?"
YES: 82% NO: 18%

Computer Stores: "Good, but expensive" was the general response.

Selected comments from "What do you think is the biggest problem facing personal computing now?":

"Information spreading", "mercenaries", "need free software", "long mailing waits", "number of software compilers, price", "software for the 6800".

IMSAI "INCOMPATIBILITY"

Dear Jim, Oct. 14, 1976

I have just entered a real-life description of "compatibility". While trying to figure out why a simple three instruction program would not work as documented in the Intel 8080A manual being executed on my IMSAI 8080, I discovered that the flag bits (as stored in memory via PUSH PSW) were not as Intel describes.

For openers, bit-5 and bit-3 are supposed to be '0'. On my IMSAI 8080 bit-3 was always '1', and bit-5 fluctuated with, as yet, no pattern sometimes being '0' and other times being '1'. At this time I played around a bit and found that the XRA A instruction did not work as documented. At this point, I contacted IMSAI.

Very quickly, I was put in contact with Mr. Bruce Holloway of IMSAI. After Bruce confirmed that strange things were happening with his IMSAI 8080, he researched the problem and reported the following: (my interpretation follows)

These IMSAI 8080's use a NEC 8080A chip instead of an Intel chip. NEC reported in a confidential letter to IMSAI some "minor" differences between their chip and Intel's. At all times, the chips were stated as being software compatible. The software differences are: (1) Flag bit-3 is always '1'; (2) Flag bit-5 is set '1' on subtract-type operations, and is reset '0' on add-type operations; (3) The CY (carry) and AC (auxiliary carry) flags are now properly set for both adds *and* subtract operations; (4) The DAA (decimal adjust) operation now works properly following either an add *or* a subtract (using flag bit-5); (5) THE AC FLAG IS NO LONGER CLEARED BY LOGICAL OPERATIONS. Additionally, Bruce mentioned that there are also some "minor" hardware differences, having to do with data on the same bus not being present at the same states as with the Intel chip (I wonder what problems this might cause?).

When I heard all of this, I informed Bruce that the fact that the AC flag is not cleared means that software written for an Intel chip would not work on the NEC chip. For example:

```
MVI  A,9
ADD  A    this forces the AC to be set
XRA  A    this is supposed to clear AC and CY
DAA       this should result in '00' but produced '06'
          with the NEC chip!!
```

At this point Bruce agreed with me since he has written similar code that would not function properly with the NEC chip. Bruce has informed me that this incompatibility was not known previously.

Well, in the span of the last three days, I have uncovered an 8080A "compatible" chip that is, for all purposes, as *incompatible* to the Intel 8080A as is the Z-80: programs can be written that will run properly on the Intel 8080A, but will run properly on the NEC 8080A, and vice-versa.

I am now waiting for the Intel 8080A IMSAI will be shipping shortly. What really disturbs me is not the imcompatibility itself, but not being informed. I don't believe that NEC should claim their chip to be "compatible", but I abhor the fact that their letter describing these differences was labeled CONFIDENTIAL and not released to the end user of their chips.

I hope that this letter may save some people untold hours debugging a program that doesn't work because of the NEC chip. Hopefully, IMSAI will refrain from using such incompatible chips on MPU boards, and will exchange customers' NEC chips for truly compatible chips, or at least distribute the NEC 'confidential' documentation.

Sincerely,

Glenn S. Tenney
Sr. Designer
(415) 574-3420

Compro
2111 Ensenada Way
San Mateo, CA. 94403

IMSAI RESPONDS

Dear Mr. Warren, Oct. 18, 1976

Following is the letter promised per our telephone conversation of October 15, 1976. We will be sending this information to all past and future customers who may have the NEC chip.

All of the features described in the following synopsis were designed by NEC to improve the 8080A chip.

Thank you for your cooperation.

Very truly yours,

IMS ASSOCIATES, INC.
Marvin Walker
General Manager

14860 Wicks Blvd.
San Leandro, CA 94577
(415) 483-2093

SUMMARY OF DIFFERENCES BETWEEN I8080A AND uPD8080A

1. During an interrupt, an RST or CALL instruction is accepted by both both processors. With the uPD8080A during M2 and M3 of a CALL instruction, the INTA status signal remains active. The I8080A requires the use of an 8228 to generate INTA by decoding 02H (all status inactive). Both I8080A and uPD8080A work correctly with Intel and NEC 8228/38.
2. Interrupt during HALT state, with the uPD8080A INTE is reset at T2.02 of the next clock period following the sampling of INT, as opposed to the I8080A where INTE is reset at M1.T1.02 of the interrupt instruction fetch.
3. Instruction Execution Times: All instruction execution times are the same except the following, which require the listed number of T (clock) states assuming no wait cycles.

	I8080A	uPD8080A
MOV r,r	5	4
RET	10	11
DAD	10	11
XTHL	18	17
SPHL	5	4

4. Data on Address Bus during M1, T4 and T5 with uPD8080A is the same as during T1-T3. With the I8080A, the Address Bus is undefined during T4 and T5.
5. Subtraction is performed as a direct binary operation in the uPD8080A and the carry, Auxiliary Carry and subtract flags are properly set to indicate the subtract operation and borrows from each four bit nibble for use with the DAA instruction.
6. DAA instruction works correctly, directly following both addition and subtraction operations with uPD8080A, while I8080A BCD subtraction must be performed by a sequence of additions and subtractions.
   With uPD8080A, three flags, Carry, Auxilliary Carry and SUB, are used for DAA operation, both for addition and subtraction (see Section 8). Carry and Auxiliary Carry are properly set to indicate borrows/carries from each four bit nibble for use with the DAA instruction. SUB flag is used to determine whether required DAA is for addition or subtraction. BCD arithmetic programs written to run on I8080A will also run on uPD8080A *unless the operations ORA, XRA, ORI, XRI, INR, DCR or DAA are depended on to affect the AC flag. Also see Section 7.**
7. Flag Registers for I8080A and uPD8080A are as follows:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
I8080A	C	1	P	0	AC	0	Z	S
uPD8080A	C	1	P	1	AC	SUB	Z	S

   Note that if the flag byte is pushed on the stack to be used as a byte in any operation such as a compare, that the value will be different for the I8080A and the uPD8080A.
8. All flags are set the same for I8080A and uPD8080A except as noted.
   A. Number of Flags:
      I8080A: Five flags
      Zero, Carry, Sign, Parity and Auxiliary Carry
      uPD8080A: Six flags
      SUB is sixth flag (subtract)

**We suggest the use of a SUB A to clear the AC and Flags, since the common XRA A does not clear the AC flag on the uPD8080A.*

SUB flag is:
set by . . . DCR, SUB, SBB, CMP, SUI, SBI and CPI
reset by . INR, ADD, ADC, ADI, ACI and DAD
affected by POP PSW

B. Affect on Flags:
Except as noted, the affect on the five common flags (Z, C, S, P and AC) are all the same for I8080A and uPD8080A.
I8080A: AC is affected by INR, DCR and DAA
AC is reset after logical operations ORA, XRA, ANI, ORI and XRI
AC is not always set correctly to indicate borrow from bit 4 after subtraction. (Subtract is performed by two's complement and only Carry is complemented to indicate correct borrow.)
uPD8080A: AC is not affected by INR, DCR and DAA
AC is not affected by logical operations
AC is always set correctly to indicate borrow from bit 4 after subtraction.

9. Status information for I8080A and uPD8080A is the same except as follows: During HALT Acknowledge, D7 (MEMR) I8080A = 1, uPD8080A = 0; during Interrupt Acknowledge while HALT, D3 (HLTA) I8080A = 1, uPA8080A = 0; and during CALL instruction following interrupt, DO (INTA) during M2 and M+ for I8080A = 0 and for uPD8080A = 1.
10. Pull-Up Resistors on the Data Bus: The uPD8080A does not utilize active pull-up resistors on the Data Bus. To make interfacing easier on the DATA BUS $V_{IN\ MIN}$ = 3.0 volts for the uPD8080A vs. 3.3 volts for the I8080A. With uPD8080A, DATA BUS input leakage current is the same as any other input.
11. The temperature range for the I8080A is 0—70 degrees C. and for the uPD8080A is -10 to +70 degrees C.
12. DC characteristics are the same except as noted:

	I8080A	uPD8080A
V_{IH}	3.3 Min.	3.0 Min.
V_{OH}	—	3.5 Min. @ I_{OH}= -1.0 ma
I_{DD} (AV)	Typ = 40	Typ = 55
I_{DD} (AV)	Max = 70	Max = 75
I_{CC} (AV)	Typ = 60	Typ = 50
I_{CC} (AV)	Max = 80	Max = 70
I_{DL}	-2.0 ma Max.	±10 ua Max.
I_{FL}	-100 ua Max.	-10 ua Max. @ V_{IN} = V_{SS} + 0.45V

13. AC characteristics are the same except as noted. See data sheet for details:

		I8080A	uPD8080A
t_{DO1}	output delay from 01 low (SYNC,DBIN)	—	160ns Max.
t_{DS2}	data setup time to 02 during DBIN	150 ns Min.	—
t_{RS01}	ready setup time to 01 high	—	240ns Min.
t_{IS}	INT set up time	During 02 for all modes except HALT mode During 01 in HALT mode	During 02 for all modes

14. All instructions are executed in the same sequence except XTHL. The uPD8080A first reads the top of the stack then writes the contents of the L register into the top of the stack, next it reads the data at the stack pointer +1, and then writes the contents of the H register into the stack pointer +1. The I8080A reads the stack twice then writes the stack twice.
15. Data on Data Bus During T4 and T5:
I8080A: The contents of the internal bus during T4 and T5 are available at the data bus.
uPD8080A: Data Bus is in the high impedence state during T4 and T5
16. HOLD Operation while DAD:
I8080A: Same timing as HOLD in Write mode, i.e., HLDA appears from 01 of the state following T3, and Address/Data Bus goes into floating state from 02 of the state following T3.
uPD8080A: Same timing as HOLD in Read mode, i.e., HLDA appears from T3-01 and Address/Data Bus goes into floating state from T3-02.

NEC RESPONDS

Dear Mr. Warren: Nov. 2, 1976

We recently received a copy of a letter sent to you by a Mr. Tenney, and feel that it is appropriate for us to respond. We hope that this will eliminate any concerns your readers may have about the use of the NEC μPD8080A.

All the differences between the Intel I8080A and the NEC μPD8080A are clearly stated in our μPD8080A Family data sheet and μCOM-8 Software Manual. These documents are available through any of our distributors, representatives or NEC Microcomputers, Inc. These documents clearly enumerate the additional features which lead to the improved performance of the μPD8080A, both in simplified code and faster execution.

However, a user need not utilize these features in his program. If the application program is written for the Intel 8080A, it will run on the NEC part except for a few very limited situations. For an example, all NEC PDA-80 and Intel MDS-800 programs operate properly using either part, as do all system programs in the IMSAI 8080 and the Altair 8800 to the best of our knowledge. It is obvious from the differences that one can create sequences of code that do operate differently in several of the 8080A's on the market today, but most of these do not represent useful sequences of application programs.

As far as the CLAIMS we make as referred to in Mr. Tenney's letter, we do claim as explained above that the μPD-8080A is "compatible", Upward Compatible! This is an improved part and we do not believe that our or IMSAI's customers should be limited to the functionability of the 8080A when an improve part is available. We are concerned that some people do not understand the advantages of the μPD-8080A. Therefore, we encourage you to print this letter for your readers.

If there is anything else we can do to help you in this matter, please contact me.

Very truly yours,

David F. Millet
Technical Staff
Microprocessors

NEC Microcomputers, Inc.
5 Militia Dr.
Lexington, MA 02173

TSK, TSK . . . OUR HEADLINE WAS ONLY OFF BY A FACTOR OF 1000

Dear Jim, Oct. 12, 1976

Many thanks for your nice article about our super duper, low priced magnetic tape storage products on Page 6 of your September issue. Only one thing wrong with it: You're guilty of overbyte in the headline. Our maximum capacity is 60 Kilobytes, not Megabytes. The text had it right. **[Must have been "a computer error." — Ed.]**

Two problems, come to think of it. Your second paragraph says that we manufacture only the drive and cartridges. Not true. In addition to the drive and cartridges, we also make what we call a digital OEM system. Unfortunately, the OEM system sells for $390 in single quantity, which is a byte much for the non-manufacturer to chew.

Yours very truly,

Irma R. Johnson
Vice President

Micro Communications Corp.
80 Bacon St.
Waltham, MA 02154

Product Review: POLY-88 -- AN EXCELLENT SYSTEM

PLUS NOTES ON SOME S–100 "GOTCHAS" -- CATCH 16 HEX

by Jef Raskin

Box 511, Brisbane CA 94005
(415) 467-4674

In our last "Gotcha" a few manufacturers were taken to task, and fewer still were praised for the quality of their products and documentation. This time we take a brief look at the familiar Altair and IMSAI chassis, and a long look at the very interesting Poly 88.

I refuse to revive the old Altair vs. IMSAI debate. As everybody now knows, the old Altair power supply was feeble. My Altair worked fine *after* I had replaced its supply with a custom 40 amp at 8 volt wonder. [In an external box that should never land on your foot.] I actually liked the Altair case better than IMSAI's, and am glad to see MITS has carried the design over to its new machine. Just two screws and the top slides off. If the screws are omitted, the case is just as strong to top loading. On the IMSAI there are four load bearing screws which I could never get at because something was always sitting alongside the computer. If the screws are omitted, the top sits rather low. Not good.

Since I get letters asking: Yes, the heavy duty supply on the IMSAI is excellent. I haven't tested the new MITS supply, but it looks good. I don't think they'll make the same mistake twice. After all, if they're smart enough to make a computer . . .

I find a serious flaw in the IMSAI front panel. Those big paddle switches that make the IMSAI look sort of like a PDP-11 have a small space between their tips. IMS should take note, as it prevents errors, and still makes it easy to hit two at once *on purpose.*

The Poly 88's almost S-100 bus (hobbyist bus, Altair bus, I could care less) has the best switches of all: there aren't any. Polymorphic Systems, in Goleta (rhymes with "Lolita") California makes this very unassuming little box that does a lot of things right, which the bigger names (with bigger boxes and price tags) are doing wrong. Not that Poly is perfect. My corrections to their manuals were extensive and numerous. But they listen harder. For example, when I called up IMS with a long list of carefully annotated errors in their manuals, they put me off, promised to call back, never did, put me off when I called again, etc. In the end, my careful documentation of their manuals did them no good at all, and frustrated me. Polymorphic Systems listened, sent me extra manuals so that I could send them mine with corrections and the like. They're not dumb; they've got a proof reader working for free. It is my opinion that manufacturers should hire a proof reader *before* sending manuals out, but the way that is now used is cheaper and only has the drawbacks of having a few hundred frustrated customers out there. And, they get hundreds of phone calls of the form, "Where does R21 go?" (Maybe the phone company is behind the bad manuals.) The Poly-88 system – which has replaced my Altair 8800 and my IMSAI 8080 – has but two controls on the box. An on-off switch with a power-on indicator light, and a reset button with a halt light. That's all you get; that's all you need. It surely doesn't look impressive. Sort of like a toaster in size and shape. The Poly is by far the easiest of the S-100 bus computers to build. The backplane and power supply are all on one well-designed motherboard. The only wires leading to it are from the transformer and the front panel button via two Molex connectors. It is all very neat with almost no point-to-point wiring. Someone was thinking when they designed this one. To take the backplane/motherboard out, just pull the two connectors and undo six easy screws (which go into captive nuts on the board, nothing to get lost inside). Have you ever tried to take out an IMSAI backplane? More screws than an X-rated movie, and then there are wires screwed onto the board. Dumb.

For some reason Poly's tiny little case requires eight screws! A bother. The kind of thinking that went into the electronic design was absent when the case was created. It is on such small shoals that great ships are wrecked. There are more bolts than slots. There are 5 slots. Is that enough? Let's see, one for the CPU, one for the video board (it's a dandy), one for a ROM board to hold something comfy, like BASIC, and, say, a 16K RAM board. With a Pixie Verter and a keyboard you're ready to go into any American household with a TV, plug in and program away. And you've a lot left over – another 16K? Voila! (or 'Cello! for that matter) we have a 32K computer with serial port, cassette interface, video interface, and software aplenty, power supply and on/off switch tucked away in our viola case. I must mention that a two-port serial interface is built into the CPU, and the RS-232 and cassette interface cards are snug against the connectors in the back of the case. That is a lot of computer in an itty-bitty base.

That isn't enough? You say you have a pile of old 4K boards? I do. Poly has the Idea of the Year (at the rate we get new ideas in this business, maybe the idea of the month): At one side of the chassis the backplane terminates in a male S-100 bus connector. On the other side of the chassis is a female connector. Aha. You can buy another chassis, put it alongside the one you already have, sort of nudge them together and guess what. Nope, you don't get a litter of 4040's. You push them together and you get . . . a ten-slot chassis. Actually, eleven slots 'cause you can put a card into the end and let it stick out instead of yet another chassis. This is useful and saves need for an extender board. The power supplies are separate and *not* bussed together. Like the IMSAI this machine has a substantial power supply. They rate it 6 amps at 5 volts. I loaded it down with 9 amps worth of boards and a length of nichrome wire and it was still putting out 8.2 volts. The point is, as you expand the chassis, you expand the power supply as well. Each expansion chassis costs $155. Takes two or three hours to build. Polymorphic Systems forgot (so typical of manuals) to tell you that R6 should be omitted on slave chassis. So I tell you.

Before I find some drawbacks (I am *not* in the employ of Poly) there was an advantage to the multiple chassis that I hadn't suspected when I ordered the miniature monsters. When working on a board, sometimes it's handy to have a program sitting around, but it disappears when you turn off the power to make a change on the board. But with the separate chassis idea, you put the CPU and memory in one box, the board under test in another. Just turn off the one chassis to remove the board, make the changes, replace and turn on the chassis. Program still there, testing continues.

Not all is peaches and cream . . . The diodes supplied with three of the four I've built–you know, the little ones for the plus and minus 16 volt supplies–were small signal diodes instead of power rectifiers. The smoke test lived up to its name. A quick trip to Radio Shack (it was Sunday) got me a handful of diodes of the right rating. 20 for $1.98 or something like that. The first one I tested was bad. But there were 21 in the bag! The other 20 were good, so no complaint there. But, dear reader, *always test.*

The assembly instructions were terrible. There were as many many errors as the other brands had [see *DDJ,* Vol. 1, No. 8].

Lots. They say they're coming out with a new manual. If it's any good, I'll probably write it up. Volume II of the instructions, however, is on the side of the Angels. This is the clearest manual on the 8080 instruction set I have ever seen. I leave it on the living room floor for people to pick up and read. It's that good. Someday I may even get a coffee table for it. So if you want to learn the 8080, get that manual. Maybe some magazine will serialize it (in good serial style: "Last month our hero got saved from the evil Dr. Halt when an interrupt arrived in the nick of time . . .").

What is life like without a front panel? Sheer joy, my friends, sheer joy. You can deposit, examine, single step, everything. You can do it in style, from a keyboard. When you single step you see not only the address and the contents (and in hex, not in binary [less than joyful to those who prefer octal]) but you also see: the accumulator, the flags, the B-C register, the D-E register, H and L, the program counter and the stack pointer.

And you also see the first eight bytes of the stack, the location the PC points to, the next seven bytes thereafter, and the eight bytes pointed to by each of the B, D and H registers. It certainly beats lights; it also beats the hex displays found on a few other machines.

I hate to say this, but the Polymorphic advertisements *understate* the advantages of their machine. Too bad for them. It should be clear that the conventional front panel is a holdover from an earlier era. It's too bad that those lights and mysterious switches appeal to so many of our computer cult. Like those famous tailfins on cars, it impresses the neighbors, but doesn't make the machine run better. Of course this goes for all the ROM replacements for front panels. Having *both* a front panel and a ROM monitor is fine; you just have to pay for it.

For just under $600 you get, with the Poly system, the bos, power supply, video board, the monitor in ROM, 512 words of RAM, room for 3K more of ROM, and all the sockets you need for the ICs. Of the S-100 bus machines, it is the only one where the minimal system has to do real programming. (O' course, you have to add a keyboard and a TV monitor – but nobody includes them for the price.) Enough free advertising for Poly. I am not so much interested in selling computers for them as I am in seeing my computer cousins not wasting their time flipping switches and misreading lights. Any system (as I said) with HEX display is better for a human being than the same system with a BINARY display, and of the S-100 systems available this week, the Poly will get more done per your hour than any of the others – that's building hour, programming hour, and even earning hour. Other manufacturers, if you've got a better system, tell me about it. Don't bother, unless you use the S-100 bus (so we can go to others than just you for add-ons). But, do tell me if you've got something *really* different like 8K for $50.

A disadvantage of not having the conventional front panel (after all this, I do know one disadvantage) is that the CROMEMCO Bytemover program won't work. It needs switches. You can get a parallel port and eight switches and wire it up for port address FF, but that's a bother. So I called up CROMEMCO (if their documentation had been better this would not have been necessary) to find out how to write a 2707 EPROM *without* their program. The method, they told me, consists of writing each PROM in its entirety, from beginning to end, a number of times. Say a hundred to three hundred times. "How many times does the Bytemover write each PROM?" I asked. "Thirty-two," I was told. So I wrote a little program that wrote the stuff into all the PROMS 255 times (you can guess why). It worked. I sold my PROM containing the Bytemover program. [Another disadvantage of no switch register is that it means the user has no sense switches, often useful in man-machine interacting programs. – Editor]

There *are* a few devilish "gotchas" in the Poly system. The first problem showed up in trying to run MITS BASIC (duly purchased from MITS). Since the monitor likes to reside in low memory, and so does BASIC, there was a conflict. The solution: Poly provides a jumper to cut, and one to add to allow you, under program control, to switch back and forth between the processor-board memory (3K ROM, ½K RAM) and other memory having the same address. In my case the program that copied BASIC from PROM resided at location E400, so I used the following program to get BASIC up:

3E 20 D3 04 C3 00 E4

The seven bytes say: put 20 (hex) into the accumulator, then send it to port 4. Sending the 20 to port 4 turns the onboard stuff off. The next three bytes of the program jump to E400 to get the program started. Now, of course, that little program is on the PROM.

The next problem was with the old MITS serial board. I've always wondered why some serial boards have a crystal-stabilized clock of their own since they could just count down from the CPU's clock signal which is on the bus. That's what the old serial board did – it counted down. PROBLEM! The clock rate on the Poly system is a few percent slower than the MITS and IMSAI clock. So the old MITS serial I/O counts down to the wrong baud rate. You have to (as I did) rewire all the counter presets. You will have to calculate the proper values. Since my computer wasn't up 'til this was fixed, I was glad to have my Model-T vintage HP-35 to do the necessary calculations. The newer MITS 88-2SIO (a fine board in my book) does it right and has its own crystal (and works like a charm in my Poly, without modification).

And another, almost unforgivable error on the Poly: it is *not quite* an S-100 system. Sure, everything I tried with it worked except for that one board. But when a manufacturer came by my place with a prototype of their new 16K board and plopped it into the Poly it didn't work. A few "unimportant" outputs and inputs to the CPU were left off the bus. More importantly the WAIT signal is not on the bus. This let the memory know that the computer was in the HALT state, which the memory needs to know. This is not the place to go into that (gotta leave something for other articles). This particular device did not need the other signals but some new board might. The point is this: you are either on a standard or not on a standard. There is no in-between. Polymorphic Systems' Poly 88 is not really an S-100 computer. You have to ask first: does the board you wish to use with it require separate disables (address, status, data out)? Does it need HLDA, INTE, WAIT? If it does, then the Poly won't do. The disables are on the bus, in non-standard locations, but they can only be disabled as a group (as required by DMA's). *One* of the output signals such as WAIT can be fed through a spare buffer on the board to the bus. (For your information it's IC 8, an 8T97. Because this is *DDJ* I'm sure the editor will permit the gory details.) Bus lines 22, 18, 23, 26, 19, 28 and 27 are not on the bus. To put WAIT on the bus, jumper pin 24 on the 8080 to pin 2 of IC 8, and pin 3 of IC 8 to bus line 27. I don't know if I've missed something else that should have been on the bus. I called Polymorphic Systems; they tell me that are putting out a list of differences, and plan to connect the WAIT as I suggest. It's the least they can do.

Another problem with the Poly is that when a number of chassis are plugged into one another, the cooling, adequate with a single system, becomes inadequate. Not only does the system *look* like a toaster, . . . My friend, Kent Strother, made a cardboard enclosure with a super quiet ROTRON fan (as per the IMSAI). Now even the regulators run cool. The secret: put the fan on top, sucking up, thus forced air aids natural

convection. There are slots around the periphery at the bottom of the case, and all other openings are sealed, forcing the air to pass the boards and transformer. Not enouth attention has been given to air flow in the IMSAI or the ALTAIR 8800 which both have a lot of stagnant air spots even with the fan going. Kent also designed a cardboard case for the keyboard that used to lie around naked. Call us the Cardboard Computer Company. It's cheap, in keeping with our homebrew budget, and if done carefully looks surprisingly good.

Second, a word for the Poly video board, but first, a word for the Processor Technology video board. The PT VDM is a top-notch piece of equipment. Their check-out procedure as you put in the chips is a classic of good manual writing. The VDM board I built worked perfectly. The Poly video board suffers from drawing current at the hairy edge of what the regulator can handle. What I like about it is the relatively fine graphics it allows: a 128 by 48 bit resolution. The graphics can be mixed with alphanumerics in any arbitrary way. It is a good use of that eighth bit that the ASCII code doesn't require. Use of the Motorola MCM6571AL character generator gives me upper and lower case Roman characters as well as the Greek alphabet and a gaggle of other special characters, including the entire official ASCII set, the square root symbol, etc.

I don't know enough to write articles like this without some help from my friends so: thanks to Doug Wyatt, my constant colleague on microcomputers, Kent Strother for the cardboard craftsmanship, Steve Calebotta for finding the problem with the missing WAIT, and out editor Jim Warren for the phrase "Hidden Gotchas" that graces these articles [who plagerized it from Dave Wyland at Ratheon]. If you find any hidden, or just plain hanging out gotchas, send them to me. I'll check them out and include them in a future article. You'll get credit in the mag, and as much moola as I get. Zilch. [Ahhh . . . but such glory and fame you get!]

PRAISE FOR RASKIN & SUGGESTIONS FOR *DDJ*

Dear Jim, Sept. 25, 1976

I just read the very informative articles by Jef Raskin ["A Bit of Wheat Amongst the Chaff"—a critique of problems found in a number of kits] in the September issue and think it is the greatest aid to the hobbyist planning on purchasing his computer system. I certainly feel that both *Dr. Dobb's* and Jef Raskin have done a great service for the hobbyist. Please publish more articles like this one. If Jef has inputs on software by all means let's hear them. Jef's appraisal was, I believe, very objective.

In order to make more room for articles of value I would like to see you eliminate as much as possible the references to new clubs and new stores which I feel are more than adequately covered elsewhere, i.e., *Byte, Interface, On Line,* etc. If you *must* how about a one-liner like *On Line.* **[These items are used only as "filler" items.]**

You might consider using smaller type for all articles in order to make room for your backlog of articles to be published. **[We're willin'. Are there any objections to dropping from 10-point type to 8-point type?]**

It might also be appropriate to eliminate items such as the article on Energy Publications which does not seem to be pertinent to the subject matter of *Dr. Dobb's.* **[This also happened to be a filler article of the right size. However, we do admit to a soft spot in our hearts for the topic of energy problems and people-oriented alternative energy sources. Our only excuse for its inclusion is that it is a technology-related subject closely associated with consumer advocacy.]**

Also, did you really need two pages for the *PCC* ad? **[*PCC* newspaper was *Dr. Dobb's* mama. Would you have us ignore our ma? Anyway, it's a "product" we think is well worth the attention of hobbyists, and as such, we published details about it.]**

All in all, I think *Dr. Dobb's* is great, but it can be better. **[We agree.]**

Very truly yours,

R. I Demrow
11 Linda Rd.
Andover, MA 01810

SWTPC KEYBOARD IMPROVEMENTS

Dear Jim,

Here's a plug for the latest in keyboards from SWTPC. For those of us whose only experience with SWTPC keyboards is with the original ones, the bad memories may make one a bit leery about giving them another chance. The new KBD-5 has much better contacts than before, and it uses the 2376 keyboard encoder which makes it a cinch to redefine keys. At $50, plus another 10 for a UART+ to make it standard serial RS-232, this is the best possible buy I can imagine.

Bob Powell
22 Bunker Hill Run
E. Brunswick, NJ 09916

A [WELL DESERVED] RAVE REVIEW FOR THE APPLE COMPUTER SYSTEM & FACTORY RESPONSIVENESS

Editor, Journal of Dentistry Oct. 11, 1976

Dear Jim,

Last July I found out about the miracles the silicon engineers were performing. I was immediately hooked on home computing. But since I had very little experience with computers, I was worried about all the delays that everyone seemed to complain about when they returned equipment for serving or clearing up the inevitable bugs.

After spending the summer carefully examining the systems available, I went to the PerCom convention in Atlantic City at the end of August. There I saw an Apple computer working. Well, it was love at first sight. I bought one from Itty Bitty in Evanston and took it home with me to New York. Well, it took them three weeks to send me two cables and two transformers and a keyboard that were needed to run it! When they finally arrived (after a prompting phone call), my Apple developed a glitch almost immediately (the only Apple that the Computer Mart of NY has ever seen with a problem!). Fearfully anticipating a two-month delay, I sent it to Steve Jobs at Apple Computer Co. He got it back into my hands in two weeks working perfectly!! He even explained an elementary error in a simple program I had tried! Since it took me over a month to even get literature from Sphere (and I wrote them a personal latter) or the Digital Group, this must be a record for a personal computer company.

My Apple is terrific. Last night I loaded your 6502 floating point routine and can now multiply, etc. Unfortunately all the answers are complemented. I've got an 8K system with a cassette storage unit, keyboard, (used) TV monitor, and a 4K BASIC (that's not quite finished yet but does run) for less than $1K. Moreover Apple promises (and I'm now a believer) to replace my current memory chips with the new 4K dynamic chips sometime next spring for about $500. The Apple was designed to be pin compatible with the new chips and so I'll have 32K on board.

Any company that can produce equipment like this and then match it with their service is really great. Companies like them and magazines like yours make home computing an accessible field for everyone.

Yours,

Raymond T. Hoobler
789 West End Ave.
New York, NY 10025

[Apple Computing is at 770 Welch Rd., Palo Alto, CA 94304, (415) 326-4248]

PRAISE FOR SUNSHINE COMPUTER CO.

Dear Editor, Nov. 24, 1976

A commendation is in order for Sunshine Computer Company of Carson, California. I bought a Sanyo cassette recorder from them on Saturday, Nov. 20. They forgot to pack in the AC cord. When I discovered the omission Saturday night, I made a note to call them Monday about it. Sunday night they called and apologized, and said they'd mail it Monday. Received it Tuesday, postmarked Monday, as promised. Total time to resolve my query: *minus* one day.

Sincerely,

Jim Raehl
943 Begonia
Escondido, CA 92027

It's a BASIC,

It's an APL . . .

It's CASUAL!

Bob Van Valzah
1140 Hickory Trail
Downer's Grove, IL 60519

Dear Jim, Sept. 12, 13, 15, & Oct. 2, 1976

Here is my entry in the hobby software field. It's a tiny language called CASUAL. That's the Chicago Area Small Users Algorithmic Language. Here are the design goals used:

- Must run on *any* 8080 system with a terminal and 2K of RAM starting at 000 000.
- Complete machine control is possible – inputting from and outputing I/O ports, memory READ and WRITE (PEEK and POKE), machine language CALL.
- 16 bit everything – line numbers, expression values.
- Arrays
- String I/O
- One tape works on any system – POKE's itself for most popular I/O boards (like MITS BASIC).
- Deletion of unwanted features at initialization time.

Note: The source tape alone is 2½ inches think (fan fold), somewhat greater than 75000 characters at this time.

I am currently at version .164. Version .09 was distributed to local hobbyists three weeks ago. That version was 200 bytes longer and had some small bugs.

While I don't have the time or the means for mass distribution of CASUAL, binary paper tapes and documentation are available from:

1. CACHE Software Library, Lloyd Smith, 530 Pierce Ave., Dyer, IN 46311
2. Chicago Computer Store, Lou Van Eperen, 517 Talcott Rd., Park Ridge, IL 60068
3. Itty Bitty Machine Company, 1316 Chicago Ave., Evanston IL 60201
4. American Microprocessor, Ed Cooper, at the Chicago Land Airport, 20 N. Milwaukee Ave., Prairie View, IL 60069.

Converting CASUAL to other CPU's:

If you're using a machine without a hardware stack, I'd say start from ground zero. CASUAL makes extensive use of the stack and almost no use of memory direct instructions. By putting CASUAL's stack on my VDM, I was able to count 48 bytes on the stack while LIFE was running. Perhaps an 8008 with hardware stack mod wouldn't be too bad, but it would take a lot more than 2K.

Memory Info: If all features are retained at initialization, 403 bytes are left in a 2K system. If all features are deleted, 610 bytes are left (slightly over ½K). The code for the interpreter, I/O drivers and all buffers except the CASUAL Program buffer takes 1.61K. If all functions are deleted, it takes 1.40K. The first 1K can be ROM or protected-after initialization.

The mnemonics for the assembler have been significantly modified from the "Intel Standard" mnemonics.

Loading Time: It takes about 5 minutes to load and initialize itself at 10 CPS. Using a Tarbell Cassette Interface at maximum baud rate, it should take about 4 seconds.

Dr. Dobb's is superb!!! Keep up the fine work!

Bob Van Valzah (312) 852-0472 (Home)
1140 Hickory Trl. (312) 971-2010 Ext. 231 (Work)
Downers Grove, IL 60519

P.S. I'm 18 years old and entering my second year working for an EE degree.

NOTES ON MY ASSEMBLER

MY OPCODE	INTEL STANDARD	
LB H	MOV	B,H
SP <> HL	XTHL	
DE <> HL	XCHG	
SP <HL	SPHL	
LXI HL LABL	LXI	H,LABL
JFZ	JNZ	
JFS	JP	
JTS	JM	
JTZ	JZ	
LCI 010	MVI	C,10
ND E	ANA	E
OR A	ORA	A
STHL	SHLD	
LDHL	LHLD	
LAI "G	MVI	A,'G'
CAL	CALL	
XR A	XRA	A
SU E	SUB	E
SB D	SUC	D
DSD	DW	
DSS	DB	
177	177Q	
0100	100	
LBUL=072	LBUL	EQU 72
NDI 300	ANI	300Q
CP M	CMP	M
* 200	ORG	200Q
INA	INR	A

NUMBERS

All numbers without leading zeros are taken as octal, all with leading zeroes are taken as decimal. "#" preceding a number causes it to be taken as hex.

REGISTER SYMBOLS

Eight bit registers are referenced with the letters A, B, C, D, H, and L. Register pairs are designated by PSW, BC, DE, HL, and SP.

OPERATORS

–	Subtraction or unary minus
+	Addition or unary plus
'	Swaps the first and second bytes of a 16 bit quantity
"	Evaluates to the ASCII equivalent of the character following it, with the eighth bit low
.	Evaluates to the address of the first byte of the instruction about to be assembled

CASUAL

GOOD AND BAD POINTS

```
16 BIT LINE NUMBERS, 1 - 65534
LINE-NUMBERED "BASIC-LIKE" TEXT EDITOR
CONTROL C (↑C) ABORTS EXECUTION AND LISTING
3 BYTE LINE OVERHEAD
IMMEDIATE EXECUTION MODE
MULTIPLE STATEMENTS PER LINE WITH COLON (":")
PRINTS LITERAL STRINGS, EXPRESSIONS, OR CHR$ FUNCTION
CRLF SUPPRESSION AVAILABLE
ONE LEVEL OF SUBROUTINE NESTING BUILT IN, EASILY EXPANDED
PEEK / POKE FUNCTIONS FOR READING OR MODIFYING MEMORY
INP / OUT FUNCTIONS FOR I/O PORT CONTROL
SINGLE AND DOUBLE BYTE ARRAYS - SINGLE DIMENSIONAL
SELECTABLE ARRAY BASE ADDRESSES FOR MULTIPLE ARRAYS
SIMPLE VARIABLES A - Z, 26 OF THEM
USER DEFINABLE FUNCTION, LIKE DEF FNA(X)
RUB-OUT TAKES BACK LAST CHARACTER TYPED
CONTROL U (↑U) ABORTS LINE BEING TYPED
72-CHARACTER LINE INPUT BUFFER
FOUR (4) RELATIONAL OPERATORS, <, =, >, # (NOT EQUAL)
NO OVERFLOW CHECKING ON MATH FUNCTIONS
15-BIT SIGNED ADD, SUBTRACT, MULTIPLY, AND DIVIDE
STANDARD EXPRESSION HIERARCHY, RELOPS THEN *,/ THEN +,-
PARENTHESIS TO ALTER HIERARCHY, NO LIMIT ON NUMBER
SPACES MAY BE INSERTED FREELY TO IMPROVE CLARITY
OPERATOR TO PROVIDE REMAINDER AFTER LAST DIVISION
OPERATOR TO PROVIDE RESULT OF LAST EXPRESSION
MACHINE-LANGUAGE CALL FUNCTION
SINGLE-CHARACTER KEYBOARD INPUT FUNCTION
RUN, AND RUN LINE NUMBER COMMANDS
LIST, AND LIST LINE NUMBER COMMANDS
NEW COMMAND CLEARS PROGRAM STORAGE AREA
NOTHING CLEARS VARIABLE VALUES, NOTHING !!
JUMP TO OPERATING SYSTEM PROVIDED
INITIALIZATION DIALOGUE A LA MITS
COMPLETE WITH BOOTSTRAP AND BINARY LOADERS (AUTO TRANSFER)
GOOD ERROR MESSAGES
RUNS IN 2K COMPLETE, > 400 BYTES LEFT IN A 2K SYSTEM
CASUAL IS SLOW
```

CASUAL DOCUMENTATION

CASUAL IS AN INTERPRETER WRITTEN IN 8080 ASSEMBLER. CASUAL IS ALSO THE NAME OF THE SYNTAX WHICH IS ACCEPTED BY THE INTERPRETER.

ONCE CASUAL HAS BEEN LOADED AND THE INITIALIZATION DIALOGUE COMPLETED, IT WILL TYPE OUT "CASUAL V .XX", WHERE XX IS THE VERSION NUMBER. THEN CASUAL WILL ENTER THE COMMAND INPUT MODE. THIS IS INDICATED BY THE PRINTING OF A PERIOD (".") AS A PROMPT CHARACTER. CASUAL IS NOW READY TO ACCEPT A LINE OF INPUT FROM THE USER. THE USER MAY BACKSPACE OVER TYPING ERRORS WITH THE RUB-OUT KEY. HE MAY ELECT TO START THE LINE OVER BY STRIKING THE CONTROL U (↑U) KEY. WHEN THE USER IS DONE WITH A LINE, HE STRIKES THE CARRIAGE RETURN KEY, TELLING CASUAL TO PROCESS THE LINE JUST TYPED. DURING LINE INPUT, ALL OTHER CONTROL CHARACTERS WILL BE IGNORED, EXCEPT CONTROL G (↑G) (BELL).

CASUAL HAS THE ABILITY TO EXECUTE COMMANDS IMMEDIATELY AFTER THEY ARE TYPED, OR TO STORE THEM AWAY FOR LATER EXECUTION AS A PROGRAM. CASUAL WILL SCAN THE INPUT LINE FOR THE FIRST NON-BLANK CHARACTER. IF THIS CHARACTER IS A NUMBER, CASUAL WILL SAVE THIS LINE IN THE CASUAL PROGRAM BUFFER IF IT IS NON-NUMERIC, CASUAL WILL ACCEPT THE LINE AS AN IMMEDIATE-MODE COMMAND, AND ATTEMPT TO EXECUTE IT.

EDITING

IF LINES ARE INPUT TO CASUAL STARTING WITH NUMBERS, THEY WILL BE EDITED INTO THE CURRENT PROGRAM IN THE PROGRAM BUFFER. LINES ARE ALWAYS STORED BY LINE NUMBER IN ASCENDING ORDER. THE INPUT:

```
30 .=65535
10 ?/ING/
20 ?/TEST/;
```

WILL BE SAVED AS:

```
10 ?/ING/
20 ?/TEST/;
30 .=65535
```

CORRECTIONS CAN BE MADE AFTER A LINE HAS BEEN ENTERED BY RETYPING THE CORRECTED LINE WITH THE SAME NUMBER AS THE BAD ONE. THE NEW LINE WILL REPLACE THE OLD ONE OF THE SAME NUMBER. NEW LINES MAY BE INSERTED BETWEEN OTHER LINES, AT THE BEGINNING, OR END OF THE PROGRAM BUFFER. THE USER SIMPLY GIVES THE NEW LINE A NUMBER BETWEEN THE NUMBERS OF THE LINES ABOVE AND BELOW IT. LINE NUMBERS MAY BE IN THE RANGE 1 TO 65534 INCLUSIVE.

THE USER MAY LOOK AT ALL OR PART OF THE PROGRAM CURRENTLY STORED IN THE CASUAL PROGRAM BUFFER BY USING THE LIST COMMAND. WHILE IN THE COMMAND MODE, TYPING "L" (CR) WILL START LISTING WITH THE LOWEST-NUMBERED LINE, AND STOP AT THE END OF THE BUFFER OR WHEN CONTROL C (↑C) IS TYPED. TYPING "LXXXXX" WILL START LISTING AT LINE XXXXX.

CASUAL STATEMENTS

THE FOLLOWING SECTION WILL COVER ALL THE STATEMENTS WHICH ARE LEGAL IN CASUAL. AS EACH STATEMENT IS PRESENTED, EXAMPLES WILL BE GIVEN OF ITS USE. IF POSSIBLE, I RECOMMEND TRYING THESE EXAMPLES AS THEY ARE ENCOUNTERED.

CASUAL HAS BASICALLY ONLY THREE (3) TYPES OF STATEMENTS: PRINT, ASSIGNMENT, AND STRING INPUT. THIS IS ONE OF THE REASONS FOR ITS INHERENT SIMPLICITY. VARIATIONS OF THESE THREE STATEMENT TYPES PROVIDE A WIDE RANGE OF FUNCTIONS.

THE PRINT STATEMENT

THE FUNCTION OF THE PRINT STATEMENT IS TO SEND DATA TO THE USER'S TERMINAL. SINCE THE WORD "PRINT" IS MORE DIFFICULT TO RECOGNIZE THAN A SINGLE CHARACTER, A QUESTION MARK ("?") IS USED TO SPECIFY THE PRINT FUNCTION. TRY THIS:

```
?10-4 (CR)
```

CASUAL WILL IMMEDIATELY PRINT:

```
 6
```

AS YOU CAN SEE, CASUAL RECOGNIZED THE "?" AS BEING A PRINT STATEMENT, EVALUATED THE FORMULA FOLLOWING IT, AND TYPED OUT ITS VALUE (IN THIS CASE 6).

OF COURSE, CASUAL CAN DO MORE THAN SUBTRACT. TRY THIS:

```
?10+4;10*4;10/4          ("*" MEANS MULTIPLY, "/" MEANS DIVIDE)
```

CASUAL WILL TYPE:

```
 14  40  2
```

NOTE THAT A SEMICOLON (";") IS USED TO SEPARATE THE FORMULAS.

NOTE TOO, THAT IN THE EXAMPLES, A SPACE IS PRINTED BEFORE AND AFTER EACH NUMBER. IF IT HAD BEEN A NEGATIVE NUMBER, THE LEADING SPACE WOULD BE A MINUS SIGN ("-").

LITERAL STRINGS MAY BE PRINTED BY ENCLOSING THEM IN SLASHES. TRY THIS:

```
?/THIS IS A CASUAL LITERAL STRING/
```

CASUAL WILL TYPE:

```
THIS IS A CASUAL LITERAL STRING
```

A COMMA (",") IN A PRINT STATEMENT CAUSES A SINGLE BYTE TAB CHARACTER TO BE SENT (011 OCTAL). A COMMA OR SEMICOLON ON THE END OF A STATEMENT WILL SUPPRESS THE CRLF AT THE END OF STATEMENT. SEE APPENDIX G FOR MORE INFORMATION.

THE ASSIGNMENT STATEMENT

NOTE: THIS SECTION IS DIVIDED INTO TWO PARTS: THE MEANING OF CHARACTERS WHEN THEY APPEAR ON THE LEFT OF AN ASSIGNMENT, AND THE MEANING ON THE RIGHT. THEY DON'T ALWAYS MEAN THE SAME THING.

LEFT SIDE

CHR.	MEANING
.	THE NUMBER OF THE NEXT LINE TO BE EXECUTED. .=350 CAUSES CASUAL TO EXECUTE LINE 350 AND CONTINUE FROM THERE. .=0 CAUSES CASUAL TO EXECUTE THE LINE AFTER THE CURRENT LINE. .=-1 CAUSES CASUAL TO STOP EXECUTION AND RETURN TO THE COMMAND MODE.

CONDITIONAL BRANCHING IS DONE LIKE THIS:

```
 =940*(X<10)
```

THE EXPRESSION ON THE RIGHT EVALUATES TO 940 IF THE CONDITION IS TRUE (X IS LESS THAN TEN). IF IT IS FALSE (X IS GREATER THAN OR EQUAL TO TEN), IT EVALUATES TO ZERO, AND THE NEXT LINE IS EXECUTED.

CHR.	MEANING
$	HAS THE SAME EFFECT AS "." (PERIOD). IN ADDITION, BEFORE CONTROL IS TRANSFERRED, THE NUMBER OF THE LINE FOLLOWING THE CURRENT LINE IS SAVED BY CASUAL. THE MOST RECENTLY SAVED VALUE IS RECALLED WHEN "$" IS FOUND ON THE RIGHT SIDE OF AN ASSIGNMENT. THIS IS USED TO BRANCH TO SUBROUTINES.

THIS CODE:

```
40 X=5:?X;:$=100
50 ?:?X+2;:$=100
60 ?/ PLUS TWO/
70 .=-1
100 ?/IS YOUR NUMBER/;
110 .=$
```

WILL PRINT:

```
 5 IS YOUR NUMBER
```

7 IS YOUR NUMBER PLUS TWO

IN LINE 40, X IS ASSIGNED THE VALUE 5, THEN THE VALUE OF X IS PRINTED. "$=100" CAUSES CASUAL TO BRANCH TO LINE 100 AND SAVE THE NEXT LINE NUMBER TO BE EXECUTED (50). WHEN THE SUBROUTINE IS FINISHED, IT RETURNS BY ".=$". IN THIS CASE "$" HAS A VALUE OF 50 SO LINE 50 IS EXECUTED. HERE, ANOTHER VALUE IS PRINTED, AND "$=100" CAUSES CASUAL TO SAVE THE NUMBER OF THE NEXT LINE (70). THIS TIME WHEN LINE 110 IS EXECUTED, "$" WILL BE EQUAL TO 70 AND CASUAL WILL CONTINUE EXECUTION AT LINE 70. NOTE THAT A ROUTINE CALLED IN THIS MANNER ALWAYS RETURNS TO THE LINE FOLLOWING THE LINE CONTAINING THE CALL TO IT.

! SETS THE MEMORY ADDRESS WHERE "PEEKING" AND "POKING" IS TO BE DONE. THE ADDRESS WILL REMAIN SET UNTIL ANOTHER "!" IS FOUND ON THE LEFT SIDE OF AN ASSIGNMENT STATEMENT. WHEN LOADED, CASUAL SETS "!=0", THEREFORE YOU MUST SET THIS ADDRESS BEFORE POKING, OR YOU MAY POKE CASUAL TO DEATH. SEE "&" AND "!" BELOW.

& IS USED TO STORE THE VALUE ON THE RIGHT OF THE ASSIGNMENT STATEMENT IN THE LAST MEMORY ADDRESS GIVEN WITH "!". THE VALUE WILL BE TRUNCATED TO 8 BITS BEFORE IT IS STORED. THIS FUNCTION IS SOMETIMES CALLED "POKING".

' IS USED TO SET THE BASE ADDRESS FOR THE SINGLE BYTE ARRAY. THE BASE ADDRESS WILL BE SET TO THE VALUE ON THE RIGHT OF THE ASSIGNMENT STATEMENT. NO VALUE SHOULD BE ASSIGNED WHICH IS LESS THAN THE NUMBER TYPED IN RESPONSE TO "MEM SIZ?"

" PERFORMS THE SAME FUNCTION AS "'", EXCEPT THAT IT SETS THE DOUBLE-BYTE ARRAY BASE ADDRESS.

↑ DEFINES THE USER-DEFINABLE FUNCTION. IT IS EXECUTABLE, THEREFORE, MORE THAN ONE FUNCTION CAN BE USED IN THE SAME PROGRAM, BUT NOT AT THE SAME TIME. THE VALUE ON THE RIGHT BECOMES THE NEW USER-DEFINABLE FUNCTION. AT THE TIME IT IS DEFINED, IT IS EVALUATED.

← SENDS THE VALUE ON THE RIGHT SIDE OF THE ASSIGNMENT STATEMENT TO THE OUTPUT PORT GIVEN BY THE LAST "@=" ASSIGNMENT. SEE "@" BELOW.

@ SETS A NEW OUTPUT PORT NUMBER. THE VALUE ON THE RIGHT IS SAVED FOR USE WITH "←" AND "&"

A - Z SETS THE VARIABLE A THRU Z TO THE VALUE ON THE RIGHT SIDE OF THE ASSIGNMENT STATEMENT. THE OLD VARIABLE VALUE IS LOST.

'[X] SETS THE X'TH ELEMENT OF THE SINGLE-BYTE ARRAY TO THE VALUE ON THE RIGHT OF THE ASSIGNMENT STATEMENT.

"[X] SETS THE X'TH ELEMENT OF THE DOUBLE-BYTE ARRAY TO THE VALUE ON THE RIGHT OF THE ASSIGNMENT STATEMENT. THE RIGHT SIDE

CHR. MEANING

HOLDS THE VALUE OF THE LINE CURRENTLY BEING EXECUTED. IF IT APPEARS IN LINE 130, IT HAS THE VALUE 130. IF IT APPEARS IN A DIRECT STATEMENT, IT HAS THE VALUE -1.

? CAUSES PROGRAM EXECUTION TO STOP FOR USER INPUT. A QUESTION MARK AND SPACE ARE PRINTED ON THE TERMINAL AS A PROMPT. THE USER INPUTS A LINE WITH A SINGLE EXPRESSION ON IT. THE VALUE OF THIS EXPRESSION IS GIVEN TO THE LEFT SIDE OF THE ASSIGNMENT. "X=?" CAUSES CASUAL TO STOP AND ACCEPT INPUT, WHICH IS THEN ASSIGNED TO THE VARIABLE X. DO NOT TYPE A QUESTION MARK IN RESPONSE TO THE INPUT PROMPT!! IF THE USER TYPES A RETURN INSTEAD OF AN EXPRESSION, CASUAL RETURNS TO COMMAND LEVEL.

$ SEE DISCUSSION FOR LEFT SIDE.

% REMAINDER AFTER LAST DIVISION. IF 20/6 IS THE LAST DIVISION PERFORMED, % WOULD BE EQUAL TO 2

! PEEK FUNCTION. TAKES ON THE VALUE OF THE CONTENTS OF THE MEMORY LOCATION ADDRESSED BY THE LAST ASSIGNMENT TO "!". SEE "!" FOR LEFT SIDE. RETURNS A VALUE 0 TO 255.

& INP FUNCTION. TAKES ON THE VALUE OF THE DATA AT THE INPUT PORT WHOSE NUMBER WAS LAST SET WITH "@=". SEE "@" FOR LEFT SIDE. RETURNS A VALUE 0 TO 255.

'[X] RETURNS THE VALUE OF THE X'TH ELEMENT OF THE SINGLE BYTE ARRAY. X MAY BE AN EXPRESSION. THE BRACKETS ARE OPTIONAL IF AND ONLY IF THEY ARE NOT NEEDED TO SEPARATE THE SUBSCRIPT FROM THE REST OF THE EXPRESSION. BRACKETS ARE MANDATORY ON THE LEFT SIDE.

"[X] SAME AS '[X] EXCEPT THIS IS THE DOUBLE BYTE ARRAY.

← TAKES ON THE VALUE OF THE LAST EXPRESSION EVALUATED. THIS INCLUDES EXPRESSIONS IN SUBSCRIPTS AND IN PARENTHESIS. (2+3)*← IS EQUAL TO 25.

@ THE USER FUNCTION. TAKES ON THE VALUE PASSED TO IT BY THE MACHINE-LANGUAGE SUBROUTINE IN DE REGISTER. INITIALLY, IT IS SET UP TO RETURN THE NUMERIC VALUE OF THE CHARACTER FOLLOWING THE "@". SEE APPENDIX F.

\ SINGLE CHARACTER INPUT FROM THE KEYBOARD. EXECUTION WILL STOP UNTIL A CHARACTER IS INPUT. NO PROMPT IS PRINTED. RETURNS A VALUE 0 TO 127. PARITY MASKED.

A - Z TAKES ON THE VALUE OF THE VARIABLE A THRU Z.

0 - 9 DIGITS OF NUMBERS INTERPRETED TO BE DECIMAL.

EXPRESSIONS

EXPRESSIONS ARE MATHEMATICAL FORMUALS WHICH EVALUATE TO 15-BIT SIGNED INTEGERS. THEY ARE USUALY FOUND ON THE RIGHT SIDE OF AN ASSIGNMENT STATEMENT, AND SEVERAL OTHER PLACES. EXPRESSIONS CONSIST OF OPERANDS WHICH GET OPERATED UPON, AND OF OPERATORS WHICH SPECIFY THE OPERATION TO BE DONE. ALL THE LEGAL OPERANDS HAVE JUST BEEN GIVEN IN THE SECTION COVERING THE RIGHT SIDE OF AN ASSIGNMENT STATEMENT.

EXPRESSIONS ARE EVALUATED USING THE STANDARD MATHEMATICAL HIERARCHY. THE ORDER OF EVALUATION MAY BE ALTERED BY USING PARENTHESIS. THIS IS A LISTING OF LEGAL OPERATORS AND THE HIERARCHY:

EVALUATED FIRST	( )
	<, >, =, #
	*, /
EVALUATED LAST	+, -

THE FOUR RELATIONAL OPERATORS EVALUATE TO EITHER A ONE (1) IF THE CONDITION IS TRUE, OR A ZERO (0) IF THE CONDITION IS FALSE. NOTE: "#" IS NOT EQUAL TO.

THE "*" AND "/" OPERATORS EVLUATE TO THE PRODUCT AND QUOTIENT OF THEIR OPERANDS RESPECTIVLY.

THE "+" AND "-" OPERATORS EVALUATE TO THE SUM AND DIFFERENCE OF THEIR OPERANDS RESPECTIVLY.

THE "+" AND "-" OPERATORS ARE ALSO USED TO INDICATE UNARY PLUS AND MINUS RESPECTIVLY. THE FUNCTION IS DETERMINED BY CONTEXT.

WHEN EXPRESSIONS APPEAR IN PRINT STATEMENTS, CARE MUST BE TAKEN TO ENSURE THE MEANING OF THE ")" AND "/" OPERATORS ARE NOT MISINTERPRETED. BOTH OF THESE CHARACTERS **DO A DOUBLE DUTY AND ARE EAISLY MISUNDERSTOOD BY CASUAL. FOR INSTANCE, "? A /MILES PER GALLON/" WILL PRODUCE ALL SORTS OF GARBAGE BECAUSE THE SLASH (/) IS TAKEN TO MEAN DIVISION AND NOT THE START OF A LITERAL STRING LIKE THE USER** WANTED. HERE IS THE FIX: "? A;/MILES PER GALLON/" HERE IT IS CLEAR THAT THE VALUE OF A IS TO BE PRINTED, FOLLOWED BY A LITERAL STRING.

ARRAY REFERENCES ARE NOT LEGAL ELEMENTS OF AN EXPRESSION WHICH IS ITSELF THE SUBSCRIPT OF AN ARRAY.

APPENDIX A

LOADING PROCEDURE

THE PURPOSE OF A BOOTSTRAP LOADER IS TO READ A LARGER BINARY LOADER INTO MEMORY WHICH IN TURN LOADS CASUAL. THE BOOTSTRAP PROGRAM MAY BE LOADED FROM THE FRONT PANEL SWITCHES OR BY USING THE SYSTEM MONITOR ROM. IT IS 21 BYTES LONG AND GOES IN VERY QUICKLY. THE BOOTSTRAP LOADER IS SO NAMED BECAUSE IT IS FREQUENTLY USED TO BRING THE SYSTEM UP AFTER A POWER-OFF CONDITION. THUS, IT IS PULLING THE SYSTEM UP BY ITS BOOTSTRAP.

THE BOOTSTRAP PRESENTED HERE IS IN A NO-CHECKSUM FORMAT, BUT IT DOES ALLOW LEADER. IT SHOULD WORK EQUALLY WELL FOR PAPER TAPE OR CASETTE INPUT. THIS BOOTSTRAP WILL LOAD A BINARY LOADER AND THEN TRANSFER CONTROL TO THE BINARY LOADER AUTOMATICALLY. THE BINARY LOADER LOADS A CHECKSUMMED-FORMAT TAPE AND ALSO TRANSFERS AUTOMATICALLY WHEN DONE LOADING. THERE ARE TWO POSSIBLE ERRORS WITH THE BINARY LOADER: CHECKSUM ERROR AND MEMORY ERROR. THE FIRST OCCURS WHEN THE CHECKSUM READ FROM THE TAPE DOES NOT MATCH THAT CALCULATED DURING LOADING BECAUSE A BYTE WAS READ FROM TAPE INCORRECTLY. A MEMORY ERROR OCCURS WHEN DATA READ FROM TAPE IS LOADED INTO MEMORY AND CAN'T BE READ BACK. THIS CAN BE CAUSED BY BAD MEMORY, PROTECTED MEMORY OR NON-EXISTENT MEMORY. WHEN EITHER ERROR OCCURS, THE LOADER STOPS READING TAPE AND ENTERS AN INFINITE LOOP. AN ASCII CHARACTER IS PUT OUT ON PORTS 1, 10, 21, AND 23 (OCTAL): AN "M" FOR MEMORY ERROR, A "C" FOR CHECKSUM ERROR. THIS CHARACTER IS ALSO STORED IN THE HIGHEST LOCATION OF THE PAGE WHERE THE BINARY LOADER RESIDES (007 377). AFTER A MEMORY ERROR, THE HL REGISTER WILL CONTAIN THE ADDRESS OF THE BAD MEMORY LOCATION.

TO READ IN A TAPE:

1. LOAD THE BOOTSTRAP FOR YOUR I/O CONFIGURATION. CHECK THE TABLE TO FIND WHICH ONE TO USE.
2. VERIFY THAT THE BOOTSTRAP IS IN MEMORY CORRECTLY.
3. EXAMINE 000 000 AND THE SET THE SENSE SWITCHES FOR THE TYPE OF I/O BOARD YOU ARE USING. LOAD THE CUSTOM I/O TABLE AT THIS TIME IF NECESSARY.
4. PLACE THE TAPE IN THE READER. IF YOU ARE USING PAPER TAPE, MAKE SURE THE BOOTSTRAP LEADER IS IN THE READER. THIS IS THE FIRST SECTION OF TAPE WITH THREE OF THE EIGHT DATA HOLES PUNCHED. IF YOU ARE USING CASETTE, LOAD AND REWIND THE TAPE AND START IT PLAYING. WAIT 15 SECONDS, AND PROCEED TO STEP 5.
5. START THE BOOTSTRAP AT LOCATION 000 000 (SPLIT OCTAL). NOTE: LOADERS G AND H START AT ADDRESSES OTHER THAN ZERO, CHECK THE TABLE.
5. WHEN THE BINARY LOADER HAS BEEN READ IN, THE ADDRESS LIGHTS WILL CHANGE AND THE TAPE SHOULD KEEP READING.

ONCE THE BINARY LOADER HAS BEEN READ IN, THE ADDRESS LIGHTS WILL DISPLAY 007 277 WHILE NORMAL LOADING IS GOING ON. IF AN ERROR OCCURS, 007 237 WILL BE DISPLAYED. IF THE TRANSFER ADDRESS WAS NOT READ PROPERLY, 007 257 WILL BE DISPALYED. IF EITHER OF THE LAST TWO CONDITIONS OCCUR, GO BACK TO STEP 1 AND RELOAD.

LOC.	A	B	C	D	E	F	G	H
000	041	041	041	041	041	041	041	041
001	302	302	302	302	302	302	302	302
002	007	007	007	007	007	007	007	007
003	061	061	061	061	061	061	061	061
004	023	023	023	023	023	023	023	023
005	000	000	000	000	000	000	000	000
006	333	333	333	333	333	333	333	333
007	001	005	000	000	006	000	020	020
010	346	346	346	346	346	346	346	346
011	001	001	001	040	001	040	001	100
012	310	310	300	310	300	310	310	310
013	333	333	333	333	333	333	333	333
014	000	004	001	001	007	001	021	021
015	275	275	275	275	275	275	275	275
016	310	310	310	310	310	310	310	310
017	055	055	055	055	055	055	055	055
020	167	167	167	167	167	167	167	167
021	300	300	300	300	300	300	300	300
022	351	351	351	351	351	351	351	351
023	003	003	003	003	003	003	003	003
024	000	000	000	000	000	000	000	000
025							076	257
026		NOTE:	LOADERS A - F		START	003	323	
027		AT 000 000.		LOADERS G & H		323	020	
030		START AT 000 025.				020	000	
031							076	323
032							021	021
033							323	076
034							020	004
035							307	323
036								020
037								307

BOOTSTRAP	I/O FORMAT
A.	CONTROL LOGIC LOW SPEED READER
B.	CONTROL LOGIC HIGH SPEED READER
C.	MITS SIOA, B, C BUT NOT REV. 0
D.	MITS REV. 0 WITH UPDATE, USE C WITHOUT UPDATE
E.	MITS ACR (AUDIO CASSETTE)
F.	MITS 88-PIO VER. 3.2 AND LATER
G.	MITS 2SIO VER. 3.2 AND LATER
H.	MITS 4PIO

NOTE: THE BINARY LOADER POKES ITSELF TO USE THE SAME DEVICE AS THE BOOTSTRAP; THEREFORE, YOU MUST USE A BOOT-STRAP OF THIS FORM, OR REWRITE THE BINARY LOADER.

TO MAKE YOUR OWN BOOTSTRAP: PUT YOUR STATUS PORT NUMBER INTO LOC. 007. PUT A MASK WHICH WILL LEAVE THE READER READY BIT INTO LOC. 011. IF READY IS ACTIVE HI: PUT 310 INTO LOC. 012. IF READY IS ACTIVE LO: PUT 300 INTO LOC. 012. PUT THE INPUT DATA PORT NUMBER INTO LOC. 014. LEAVE ALL OTHER LOCATIONS THE SAME AS LOADER A.

APPENDIX B

INITIALIZATION DIALOGUE

AFTER CASUAL HAS BEEN LOADED (PER INSTRUCTIONS IN APPENDIX A) AND ANY I/O PATCHES HAVE BEEN MADE, IT WILL ASK:

MEM SIZ?

IF YOU TYPE A CARRIAGE RETURN, CASUAL WILL USE ALL THE CONTIGUOUS MEMORY UPWARDS FROM ZERO THAT IT CAN FIND. CASUAL WILL STOP SEARCHING WHEN IT FINDS ONE BYTE OF ROM OR NON-EXISTENT MEMORY, I.E. MEMORY WHICH WILL NOT ACCEPT AND SUC-CESSFULLY READ BACK A TEST BYTE. THIS IS A NON-DESTRUCTIVE TEST SO I/O PATCHES AND SUCH WON'T BE DESTROYED.

IF YOU WISH TO ALLOCATE ONLY PART OF YOUR COMPUTER'S MEMORY TO CASUAL, TYPE THE DECIMAL ADDRESS OF THE FIRST BYTE WHICH CASUAL IS NOT TO USE. THIS MIGHT BE DONE, FOR EXAMPLE, IF YOU WERE USING PART OF MEMORY FOR MACHINE LANGUAGE SUBROUTINES OR TO SET ASIDE MEMORY FOR CASUAL ARRAY STORAGE.

THERE ARE 4096 BYTES IN A 4K SYSTEM, 8192 IN AN 8K SYSTEM, 2048 IN A 2K SYSTEM.

THE ADDRESS GIVEN IN RESPONSE TO "MEM SIZ?" MUST BE RAM, OR ELSE CASUAL WILL REPEAT THE QUESTION. THERE IS ALSO A CERTAIN MINIMUM AMOUNT OF MEMORY CASUAL MUST HAVE TO OPERATE. IF THE RESPONSE IS LESS THAN THIS MINIMUM, CASUAL WILL REPEAT THE MEMORY SIZE QUESTION. IN VERSION .16, THIS MINIMUM IS ABOUT 1700.

CASUAL WILL NOW ENTER A DIALOG WHICH ALLOWS YOU TO DELETE SOME COMMANDS AND FEATURES. IF FEATURES ARE DELETED, THIS WILL LEAVE MORE FOR YOUR PROGRAM. HOWEVER, ATTEMPTING TO ACCES THESE FEATURES WILL GIVE AN ERROR, USUALLY THE SYNTAX ERROR. THE ONLY TO RESTORE A FEATURE WHICH HAS BEEN DELETED IS TO RELOAD CASUAL.

THIS IS THE DIALOG WHICH WILL OCCUR:

WANT SAVE/TAPE?	ANSWER "Y" TO RETAIN SAVE AND TAPE COMMANDS. IF YOU ANSWER "N", ASKS NEXT QUESTION.
WANT STR I/O?	ANSWER "Y" TO RETAIN STRING INPUT AND OUTPUT. IF YOU ANSWER "N", ASKS NEXT QUESTION.
WANT ARRAYS?	ANSWER "Y" TO RETAIN SINGLE AND DOUBLE BYTE ARRAYS. IF YOU ANSWER "N", BOTH ARRAYS ARE DELETED.

ONCE THIS DIALOGUE IS COMPLETE, CASUAL TYPES OUT:

XXXXX BYTES FREE CASUAL V. YY	XXXXX IS THE NUMBER OF BYTES AVAILABLE FOR PROGRAM STORAGE AND STACK SPACE. YY IS THE CURRENT VERSION NUMBER.

DELETING SAVE AND TAPE COMMANDS WILL FREE UP AN ADDITIONAL 106 BYTES, STRING I/O ANOTHER 34, AND DELETEING ARRAYS GIVES ANOTHER 62 BYTES.

APPENDIX C

ERROR MESSAGES

WHEN AN ERROR OCCURS, CASUAL RETURNS TO COMMAND LEVEL AND TYPES THE PROMPT PERIOD ".". VARIABLE VALUES AND THE CASUAL PROGRAM REMAIN INTACT. AFTER THE ERROR HAS BEEN CORRECTED, EXECUTION MAY BE CONTINUED WITH NO LOSS OF CONTEXT.

WHEN AN ERROR OCCURS IN A DIRECT STATEMENT, NO LINE NUMBER IS PRINTED.

FORMAT OF ERROR MESSAGES:

DIRECT STATEMENT	ERROR XXX LL?LL
INDIRECT STATEMENT	ERROR XXX IN YYYYY LL?LL

IN BOTH CASES, "XXX" IS THE ERROR NUMBER. "LL?LL" IS THE STATEMENT IN WHICH THE ERROR OCCURRED. A QUESTION MARK IS INSERTED AT THE POINT OF THE ERROR SOMETIMES. THE "YYYYY" WILL BE THE LINE NUMBER WHERE THE ERROR OCCURRED FOR THE INDIRECT STATEMENT.

THE FOLLOWING ARE THE KNOWN ERROR NUMBERS AND THEIR MEANINGS:

98 SYNTAX ERROR. MISSING PARENTHESIS, ILLEGAL CHARACTER IN A STATEMENT, OR UNRECOGNIZABLE STATEMENT TYPE.

291 ILLEGAL CHARACTER TERMINATING A STATEMENT. FOR EXAMPLE: X=3) GIVES ERROR 291.

346 MISSING CLOSING SLASH IN A LITERAL STRING.

410 UNDEFINED STATEMENT. AN ATTEMPT WAS MADE TO BRANCH TO A LINE NUMBER WHICH DOES NOT EXIST. THIS ERROR MAY OCCUR IN THE RUN XXXXX COMMAND, WHERE XXXXX DOES NOT EXIST.

516 OUT OF MEMORY. PROGRAM TOO LARGE OR TOO COMPLICATED AN EXPRESSION OR A COMBINATION OF BOTH. SEE APPENDIX D.

761 DIVISION BY ZERO.

801 MISSING EXPRESSION. A STATEMENT TERMINATOR WAS FOUND WHERE AN EXPRESSION WAS EXPECTED.

APPENDIX D

SPACE HINTS

IN ORDER TO MAKE YOUR PROGRAM SMALL AND SAVE SPACE, THE FOLLOWING HINTS MAY BE HELPFUL.

1) USE MULTIPLE STATEMENTS PER LINE. THERE IS A SMALL AMOUNT OF OVERHEAD (3 BYTES) ASSOCIATED WITH EACH LINE IN THE CASUAL PROGRAM. TWO OF THE BYTES CONTAIN THE LINE NUMBER IN BINARY. THIS MEANS THAT NO MATTER HOW MANY DIGITS YOU HAVE IN YOUR LINE NUMBER, IT TAKES THE SAME AMOUNT OF SPACE. PUTTING AS MANY STATEMENTS AS POSSIBLE ON A LINE WILL REDUCE THE NUMBER OF BYTES USED BY YOUR PROGRAM.

2) DELETE ALL UNNECESSARY SPACES FROM YOUR PROGRAM. SPACES ARE ALLOWED ON THE RIGHT SIDE OF A CASUAL STATEMENT FOR CLARITY, BUT THEY ARE IGNORED. NOTE: ALL SPACES BETWEEN THE LINE NUMBER AND THE FIRST NON-BLANK CHARACTER ARE IGNORED.

3) DELETE ALL REMARKS FROM THE PROGRAM.

4) USE VARIABLES INSTEAD OF CONSTANTS.

5) THE LAST STATEMENT OF A PROGRAM NEED NOT BE AN END STATEMENT. CASUAL WILL RETURN TO COMMAND MODE AUTOMATICALLY IF IT RUNS OUT OF PROGRAM TO EXECUTE.

6) USE SUBROUTINES TO EXECUTE SECTIONS OF CODE WHICH APPEAR IN A PROGRAM MORE THAN ONCE.

7) USE RELATIONAL OPERATORS INSETAD OF GOTOS. FOR INSTANCE: IF YOU WANT X = 10 IF Y = 10, AND X=0 IF Y # 10 : DO IT LIKE THIS: X=Y#10.

8) USE THE "←" OPERATOR INSTEAD OF REPEATING AN EXPRESSION.

STORAGE ALLOCATION INFORMATION

THE USER-DEFINED FUNCTION USES NO MEMORY TO STORE THE DEFINITION.

WHILE A PROGRAM IS BEING EXECUTED, SPACE IS ALLOCATED ON THE STACK. EACH LEVEL OF PARENTHESIS ENCOUNTERED IN AN EXPRESSION TAKES 8 BYTES OF STACK SPACE.

APPENDIX E

BASIC TO CASUAL STATEMENT CROSS REFERENCE

BASIC	CASUAL
RUN	R
LIST	L
NEW	N
350 DEF FNA(X)=X*X+Y*Y	350 ↑=X*X+Y*Y
999 END	999 .=-1
50 GOTO 100	50 .=100
10 GOSUB 910	10 $=910
16 IF X+10 > Y/2 THEN 214	16 .=214 * (X+10 > Y/2)
20 IF X>3 AND X<10 THEN 250	20 .=250 * (X > 3) * (X < 10)
140 INPUT X	140 X=?
145 INPUT Y,Z,A	145 Y=?: Z=?: A=?
147 LET A = B = 0	147 A=0: B=0
107 LET W = (2+3)*4	107 W=(2+3)*4
100 ON I GOTO 10,20,30,40	100 .=I*10*(I > 0)*(I < 5)
105 ON SGN (X)+2 GOTO 40,50,60	105 .=50+10*((X > 0)-(X < 0))
110 ON I GOSUB 50,60	110 $=40+I*10*(I > 0)*(I < 3)
355 OUT I,J	355 @=I: ←=J
357 POKE I,J	357 !=I: &=J
360 PRINT X,Y;Z	360 ?X,Y;Z
370 PRINT	370 ?
380 PRINT X,Y;	380 ?X,Y;
390 PRINT "I THINK IT'S";A	390 ?/I THINK IT'S/;A
400 PRINT A,B,	400 ?A,B,
410 PRINT CHR$(Z-INT(Z/64)*64);	410 ?>Z-Z/64*64
500 REM SMALL IS GREAT !!	500 .=510 SMALL IS GREAT !!
50 RETURN	50 .=$
9000 STOP	9000 .=-1

APPENDIX F

BASIC TO CASUAL FUNCTION CROSS REFERENCE

THE FOLLOWING TABLE CAN BE USED TO MAKE CASUAL'S USER-DEFINABLE FUNCTION EQUIVALENT TO THE CORRESPONDING INTRINSIC FUNCTION OF BASIC.

BASIC	CASUAL
ABS (X)	↑=X*((X>0)-(X<0))
SGN (X)	↑=(X>0)-(X<0)
USR (X)	↑=@
PEEK (X)	!=X: ↑=!
INP (X)	@=X: ↑=&
MOD (X)	↑=X/Y*0+%
MAX (X,Y)	↑=(X>Y)*X+(X-1<Y)*Y
MIN (X,Y)	↑=(X<Y)*X+(Y-1<X)*Y

OTHER USEFUL FUNCTIONS

OCTAL TO DECIMAL: THIS FUNCTION ACCEPTS A THREE DIGIT OCTAL NUMBER IN DECIMAL PRINT FORMAT IN THE VARIABLE C.

```
10 ↑=C/100*64+(C-C/100*100)/10*8+C-C/10*10
20 C= 377: ?↑            PRINTS 255
```

DECIMAL TO OCTAL. THIS FUNCTION ACCEPTS A DECIMAL NUMBER (0 - 255) IN THE VARIABLE D, AND RETURNS AN OCTAL REPRESENTATION OF IT AS A DECIMAL NUMBER 0 - 377.

```
30 ↑=D/64*100+(D-D/64*64)/8*10+D-D/8*8
40 D=255: ?↑             GIVES 377
```

APPENDIX G

CASUAL/MACHINE LANGUAGE INTERFACE

CASUAL HAS THE ABILITY TO LINK TO MACHINE-LANGUAGE SUBROUTINES, AND RECEIVE DATA FROM THEM. FIRST, YOU MUST SET ASIDE ENOUGH MEMORY TO HOLD THE MACHINE-LANGUAGE ROUTINE. WHEN CASUAL ASKS "MEM SIZ?", DON'T TYPE A RETURN, BECAUSE CASUAL WOULD USE ALL THE MEMORY IT COULD FIND, AND NONE WILL BE LEFT FOR YOUR MACHINE-LANGUAGE ROUTINE.

YOU SHOULD NOT ATTEMPT TO USE ANY MEMORY BETWEEN LOCATION ZERO AND THE LAST ADDRESS ALLOCATED FOR CASUAL, AS IT IS CONSTANTLY BEING MODIFIED BY CASUAL.

SINCE CASUAL MUST USE CONTIGUOUS BLOCKS OF MEMORY STARTING AT ZERO, IT IS BEST TO RESERVE HIGH LOCATIONS IN MEMORY FOR YOUR SUBROUTINES.

FOR EXAMPLE, IF YOU HAVE A 3K SYSTEM, THERE ARE 3072 BYTES IN YOUR MACHINE (1024 * 3). THEY ARE NUMBERED 0 - 3071. IF YOU WANTED TO USE A 50 BYTE SUBROUTINE, YOU WOULD TYPE 3022 IN RESPONSE TO "MEM SIZ?" THIS WILL ALLOCATE LOCATIONS 0 - 3021 FOR CASUAL, AND 3022 - 3071 FOR YOUR SUBROUTINE.

THE STARTING ADDRESS OF YOUR ROUTINE MUST BE STORED IN A LOCATION KNOWN AS "USRL". THE ADDRESS OF USRL IS FOUND AT ADDRESS 000 003 , SPLIT OCTAL.

WHEN LOADED, USRL CONTAINS THE ADDRESS OF A ROUTINE TO RETURN THE NUMERIC VALUE OF THE ASCII CHARACTER FOLLOWING THE "@". USRL CONTAINS THE TWO BYTE ABSOLUTE ADDRESS CASUAL CALLS WHEN IT ENCOUNTERS AN AT SIGN ("@") IN AN EXPRESSION.

WHEN YOUR ROUTINE IS CALLED, THE STACK POINTER IS SET UP AND YOU ARE ALLOWED TO USE UP TO 11 LEVELS OF STACK SPACE (22 BYTES). TO USE MORE, YOU'LL HAVE TO SAVE CASUAL'S STACK POINTER AND SET YOUR OWN. YOU MAY USE ALL OF THE CPU REGISTERS EXCEPT HL. HL CONTAINS THE ADDRESS OF THE CHARACTER FOLLOWING THE "@". THE RESULT OF THE @ FUNCTION IS PASSED BACK TO CASUAL IN THE DE REGISTER AS A 15 BIT SIGNED NUMBER. THE MOST SIGNIFICANT BITS ARE IN THE D REGISTER.

YOU MAY RECIEVE ARGUMENTS PASSED TO YOUR ROUTINE BY CALLING A ROUTINE CALLED SUBS. THE ADDRESS OF THIS ROUTINE IS HELD IN LOCATIONS 5 AND 6. THE ARGUMENT SHOULD BE ENCLOSED IN BRACKETS ("[" AND "]").

THE USERS ROUTINE MAY ENABLE INTERRUPTS, AS LONG AS THE USER USES ONLY RST 7 INTERRUPTS. INTERRUPTING TO OTHER LOCATIONS WILL CAUSE TROUBLE. THREE BYTES HAVE BEEN LEFT AT LOCATION 56 DECIMAL, 70 OCTAL, 38 HEX. THESE LOCATIONS ARE LEFT SO THE USER CAN INSERT A JUMP TO AN INTERRUPT SERVICE ROUTINE.

CARE MUST BE TAKEN IN INTERRUPT SERVICE ROUTINES TO SAVE ALL OF THE CPU'S REGISTERS.

DON'T FORGET TO ENABLE INTERRUPTS BEFORE RETURNING, OR YOUR MACHINE WILL NEVER SEE ANOTHER INTERRUPT.

SUPPOSE YOU HAVE A 2K COMPUTER, AND NEED A ROUTINE TO READ THE NUMBER ON THE FRONT PANEL SWITCHES.
NOTE: THIS FUNCTION CAN BE DONE DIRECTLY IN CASUAL. YOU HAVE 2048 BYTES OF MEMORY MINUS 6 BYTES FOR THE ROUTINE LEAVES 2042 FOR CASUAL. THIS IS THE NUMBER YOU WOULD TYPE IN RESPONSE TO "MEMORY SIZE?". LOAD THIS INTO MEMORY,

```
LOC.    DATA    OPCODE
007 372 333     IN      377
007 373 377
007 374 137     MOV     E,A
007 375 026     MVI     D,0
007 376 000
007 377 311     RET
```

APPENDIX H

ASCII CHARACTER CODES (DECIMAL)

#	CHR.	#	CHR.	#	CHR.	#	CHR.
32	(SPACE)	33	!	34	"	35	#
36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+
44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3
52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;
60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C
68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K
76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S
84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[
92	\	93	]	94	↑	95	←

NOTE: SOME TERMINALS PRINT CODE 95 AS A BACK ARROW, AND SOME PRINT AN UNDERLINE.

```
        CONTROL FUNCTIONS
        -----------------

LINE FEED = 10          TAB = 11                FORM FEED = 12
CARRIAGE RETURN = 13    BELL = 7

        THESE CODES ARE USED WITH THE ">" FUNCTION OF THE "?"
STATEMENT.  ">" FOLLOWED BY AN EXPRESSION RETURNS A ONE
CHARACTER STRING WHICH CONTAINS THE ASCII EQUIVALENT
OF THE EXPRESSION.
        APPENDIX Z
        ----------

        THANKS TO:

THE FOLLOWING PEOPLE FOR DEBUGGING EARLY VERSIONS:
BILL SAINDON, AL BAKER, MR. ZIEGLER, AND SEVERAL OTHER
CACHE MEMBERS.

THE FOLLOWING PEOPLE FOR PROOFREADING:
BILL PRECHT, MARK DAVISON AND MY FAMILY.

GARRY SHANNON FOR PROPOSING THE SYNTAX.

BYTE MAGAZINE FOR THE DECIMAL PRINT ROUTINE.

LOU VAN EPREN OF THE CHICAGO COMPUTER STORE FOR THE USE OF
HIS EQUIPMENT.
```

PROGRAM NAME: CASUAL PATCH SHEET

An optional patch to replace the RUN command with a CLEAR command. This command allows the user to allocate more or less memory for CASUAL after initialization. The argument is an expression which is the first location that CASUAL is not to use. This location must be RAM, and must be $>= 1703_{10}$. Returns to command mode when done. To run programs, you'll have to type .= <EXPR >, where <EXPR> is the first line number to be executed.

LOCATION		DATA		OPCODE	
HI	LOW	OLD	NEW	OLD	NEW
004	037	317	317	TST"R	TST"C
004	040	122	103		
004	041	173	173	IFNOT	IFNOT
004	042	005	005	OS?	OS?
004	043	312	337		
004	044	024	353	JTZ	EXPR
004	045	004	042	RSSP	DE<>HL
004	046	337	377		STHL
004	047	303	004	EXPR	STRS
004	050	207	307	JMP GOTA	

RAW ROCKWELL RUMORS REVEAL ZILOG COMPETITOR 9/21/76

Rockwell International had been considered by Zilog as a potential second source for the Z80 parts. Since Rockwell was not chosen, they went ahead with the development of their own Z80-type CPU chip (R80). It will be ready sometime in 1977.

The words is that the chip will be pin compatible and software compatible. The device is mask microprogrammable, similar to Western Digital's 16-bit CPU chip set. Some of the instructions execute much faster than Zilog's Z80, e.g., block move Z80 in 21 cycles/byte, vs. R80 in 5 cycles/byte. Finally, there will be some instruction enhancements over the Z80 which include a hardware multiply and divide.

MOSTEK AND FAIRCHILD TO SECOND SOURCE EACH OTHER

Mostek is already the second source for Fairchild's F-8 microprocessor chips. Now, Fairchild will become the second source for Mostek's 4K-bit fast RAM, the 16-pin MK4027 that runs at less than 200 ns.

Dear Jim, Oct. 16, 1976

I am still attempting to locate more information on a Chicago computer-fest.

Scott Meaden (a local CASUAL user) has just written a neat game in CASUAL. It's called "Zapp the Moonman". It runs with a VDM by "poking" the screen, a moving target (a moonman) moves back and forth across the top of the screen. You shoot lightning bolts at him from your gun at the bottom. I am now twisting his arm to finalize it and send you a copy.

Sincerely,

Bob Van Valzah 1140 Hickory Trl
(312) 852-0472 Downers Grove, IL 60515
(312) 971-2010 x231

ZAPP THE MOON MAN

Dear Dr. Dobbs, Oct. 21, 1976

I've been using a nifty little language called "CASUAL" on my IMSAI (which is only 4K smart at the moment) with a VDM and came up with a program you might consider for publication. I call it "Zapp the Moon Man". It starts a moonman (VDM character 7) moving from left to right and then back again across the top of the screen. Your job is to try to zap him with a lightning bolt (VDM character 4) that moves from the bottom of the screen to the top. The bolts are fired by the sense switches on the front panel of my IMSAI. The program also keeps track of how many bolts you have left and how many moonmen you've zapped.

The start of the VDM screen must be set in line 3. My VDM is set up for CC(hex) 314(octal) which, in CASUAL decimal notation, is equal to −13312. You can use CASUAL to help you figure out what number to set Z to. For example, if your screen starts on page 364 (octal), type in ?256*(3*64+6*8+4), hit return and CASUAL will print −3072. This is Z for that system. Also, if you want to use a different trigger source (say, passing your hand over an OP-80 papertape reader) you'll have to change lines 11 and 110 for port numbers and Ready bit. But remember, once that bit is ready you'll have to give the data port an & to clear it.

The game is set up so when a bolt gets to the top of the screen it checks 2 places to see if it hit (line 210). To make him harder to hit, type in:

```
210.=240*(((X-2=P)*(B=0))+((X+2=P)*(B=1)))
```

Happy zapping!

Scott Meadow 116 Surrey Dr.
Glen Ellyn, IL 60137

```
.L

3 Z=-13312
4 ?>12
5 ?/HOW MANY BOLTS DO YOU WANT/;:S=?:.=5*(S<1)
6 .=5*(S>99)
10 U=Z+754:L=Z+864:M=L:P=Z+32:N=Z-32:J=Z+722
11 @=255
15 ?>12:!=Z:&=32:H=0:I=0:!=L+128:&=160:L=L+128
20 '=Z+592:'[1]=90:'[2]=65:'[3]=80:'[4]=83
30 '[32]=66:'[33]=79:'[34]=76:'[35]=84:'[36]=83
40 '[97]=76:'[98]=69:'[99]=70:'[100]=84:S=S+1:$=400
50 B=0:X=Z:Y=Z+64
100 $=3000
110 .=100*(0=&)
120 $=400
130 !=L+64:&=32:!=L:&=4:L=L-64:$=3000
150 .=130*(L#N)
200 !=P:&=32
210 .=240*((((X-1=P)+(X-2=P))*(B=0))+(((X+1=P)+(X+2=P))*(B=1)))
220 .=5*(S<1)
230 .=110
240 H=H+1:I=H/10:!=J+1:&=%+48:!=J:&=I+48:I=H
242 '=Z+32:'[1]=90:'[2]=65:'[3]=80:'[4]=80:'[5]=33
244 X=Z:!=X:W=0
245 &=32:X=X+1:!=X:W=W+1:.=245*(W#50)
250 .=50
400 S=S-1:L=M:V=S/10:!=U+1:&=%+48:!=U:&=V+48:V=S:.=$
3000 .=3900*(B=1)
3100 !=X-1:&=32:!=X:&=7:X=X+1:.=$*(X#Y)
3200 X=X-1:B=1:.=$
3900 !=X+1:&=32:!=X:&=7:X=X-1:.=$*(X#Z)
4000 X=X+1:B=0:.=$
```

```
                        /ABBREVIATIONS USED IN COMMENTS:
                        /
                        /->             INTO
                        /ABS( )         ABSOLUTE VALUE OF ( )
                        /ADR            ADDRESS
                        /ARG            ARGUMENT
                        /BUF            BUFFER
                        /BOTX           BEGINNING OF TEXT
                        /CR             CARRIAGE RETURN
                        /CRLF           CARRIAGE RETURN, LINE FEED
                        /CHR            CHARACTER
                        /CMPR           COMPARE
                        /DECR           DECREMENT
                        /EOP            END OF PROGRAM
                        /EXPR           EXPRESSION
                        /EOS            END OF STATEMENT, OR END OF STRING
                        /EOTX           END OF TEXT
                        /EOB            END OF BUFFER
                        /EOL            END OF LINE
                        /FC             FALSE CARRY
                        /FUN            FUNCTION
                        /FZ             FALSE ZERO
                        /INIT           INITIALIZE
                        /INFO           INFORMATION
                        /INCR           INCREMENT
                        /INST           INSTRUCTION
                        /INP            INPUT
                        /LANG           LANGUAGE
                        /LF             LINE FEED
                        /LINE #         LINE NUMBER
                        /LL             LINE LENGTH
                        /NEOTX          NEW END OF TEXT
                        /OEOTX          OLD END OF TEXT
                        /OP             OPERATOR
                        /OS             OPERATING SYSTEM
                        /PS             PARTIAL SUM
                        /PGM            PROGRAM
                        /QUO            QUOTIENT
                        /RETADR         RETURN ADDRESS
                        /RELOP          RELATIONAL OPPERATOR ( <, >, =, # )
                        /REG            REGISTER
                        /ROT            ROUTINE
                        /STK            STACK
                        /STMT           STATEMENT
                        /SONL           START OF NEXT LINE
                        /SO>L           START OF GREATER THAN LINE
                        /SO<L           START OF LESS THAN LINE
                        /SOL            START OF LINE
                        /SUB            SUBTRACT
                        /SOS            START OF STATEMENT
                        /SR             SUBROUTINE
                        /SIG DIG        SIGNIFICANT DIGIT
                        /TXA            TEXT ADDRESS POINTER
                        /TST            TEST
                        /TZ             TRUE ZERO
                        /VAL            VALUE
                        /VAR            VARIABLE
                        /VARNAM         VARIABLE NAME
                        /
                        /                                               EP 1
                        /
                        /               C A S U A L
                        /
                        /  CHICAGO AREA SMALL USERS ALGORITHMIC LANGUAGE
                        /
                        /  WRITTEN BY:  ROBERT A. VAN VALZAH
                        /               1140 HICKORY TRAIL
                        /               DOWNERS GROVE,  IL.
                        /                               60515
                        /
                        /               H  (312) 852-0472
                        /               W  (312) 971-2010  X 227
                        /
000 000 303 154 006             JMP INIT        /WILL BE POKED TO JMP ENTR
000 003 007 005                 DSD USRL        /ADR OF ADR OF ADR OF USERS ML ROT
000 005 272 005                 DSD SUBS        /ADR OF ROT USED TO GET USR FUN ARG
                        /
                        /RESTART SUBROUTINES.  0 IS SYSTEM RE-ENTRY.  7 IS OPEN.
                        /1 - 6 ARE USED.
                        /
                        /THE EQUIVALENCES BELOW SUBVERT MY ASSEMBLER INTO ASSEMBLING
                        /SINGLE BYTE CALL INSTRUCTIONS (RESTARTS) WHENEVER THEY
                        /APPEAR ON A LINE.
                        /
            317 000     TST=@3172
                  327   FETCH=RST @200
            000 000     IFNOT=02
                  367   CMPR=RST @600
                  337   EXPR=RST @300
                  347   DEVO=RST @400
                  357   MSGP=RST @500
                        /
                        /RESTART 1 IS THE TST FUNCTION.  IN SOURCE CODE IT MUST
                        /BE FOLLOWED BY AN IFNOT PSEUDO - OP.  IT APPEARS LIKE THIS:
                        /
                        /       TST "+
                        /       IFNOT TRY-      /CHR AT HL IS NOT "+", JUMP TO TRY-
                        /       /FALLS THROUGH TO HERE IF CHR AT HL IS "+"
                        /
                        /THE CHR FOLLOWING THE RESTART INSTRUCTION IS FETCHED AND
                        /COMPARED TO THE CHR POINTED TO BY HL.  IF THE
                        /TEST IS TRUE, THE IFNOT ADDRESS IS IGNORED AND TST RETURNS.
                        /ALSO HL IS BUMPED AND IT FALLS THROUGH TO NXTC TO SET FLAGS
                        /IF THE TEST IS FALSE, THE RETURN ADDRESS ON THE STACK
                        /IS IGNORED AND THE IFNOT ADDRESS IS RETURNED TO,
                        /WITHOUT BUMPING HL.
                        /STACK USAGE:  2 BYTES.  MUNCHES A & FLAGS.
                        /
            010 000             *10
000 010 176                     LA M            /FETCH TEST CHR
000 011 343                     SP<>HL          /TXA ON STK, REFERENCE ADR -> HL
000 012 276                     CP M            /COMPARE WITH REFERENCE
000 013 043                     INX HL          /MOVE RETADR
000 014 303 113 000             JMP TST1        /CONTINUES AT TST1
                        /
                        /RESTART 2 IS THE FETCH THE NEXT CHR ROUTINE.  HL IS BUMPED
                        /BEFORE THE FETCH.  SPACES ARE IGNORED.  ON RETURN:  FC
                        /MEANS NON-NUMERIC (NOT 0 - 9), TZ IF A STATEMENT TERMINATOR
                        /(COLON OR END OF LINE NULL).  STACK USAGE:  2 BYTES.
                        /
            020 000             *20
000 020 043             NXTC,   INX HL          /BUMP TXA TO NEXT CHR
000 021 176                     LA M            /FETCH IT
000 022 376 072                 CPI "9+1        /IS IT 0 - 9
000 024 320                     RFC             />9  CARRY FALSE
000 025 303 100 000             JMP NXTD        /CONTINUED AT NXTD
                        /
                        /RESTART 3 IS THE EXPRESSION EVALUATOR.  THE VALUE IS
                        /RETURNED IN THE DE REG.  SEE CONTINUATION FOR MORE INFO.
                        /
            030 000             *30
000 030 315 107 002             CAL EXPA        /GET THE VALUE OF EXPR -> DE
000 033 353                     DE<>HL          /RESULT -> HL, TXA -> DE
000 034 303 073 000             JMP EXP1        /CONTINUED AT EXP1
                        /
                        /RESTART 4 IS THE DEVO (DEVICE OUTPUT) ROUTINE.  THE CHR
                        /IN THE A REG IS SENT TO THE OUTPUT DEVICE.  DOESN'T
                        /MUNCH ANY REGS OR FLAGS.  STACK USAGE:  4 BYTES.
                        /
            040 000             *40
000 040 365                     PSHX PSW        /SAVE A AND FLAGS
000 041 333 001         DEVQ,   INP 1           /GET READY STATUS -> A
000 043 346 002         TORM,   NDI 2           /MASK TO THE BIT WE WANT
000 045 303 106 005             JMP DEVP        /CONTINUED AT DEVP
                        /
                        /RESTART 5 IS THE MESSAGE PRINTER.  IT SENDS CHRS
                        /FROM MEMORY IMMEDIATLY FOLLOWING THE CALL TO IT UNTIL ONE
                        /WITH BIT 7 HI COMES ALONG.  THE RETURN ADDRESS IS
                        /MODIFIED.  STACK USAGE:  6 BYTES.
                        /
            050 000             *50
000 050 343             MSG,    SP<>HL          /PRINT ADR -> HL
000 051 176             MSG1,   LA M            /FETCH A CHR
000 052 347                     DEVO            /SEND IT
000 053 043                     INX HL          /BUMP TXA AND RETURN ADDRESS
000 054 267                     OR A            /BIT 7 HI YET ?
000 055 303 132 000             JMP MSG2        /CONTINUED AT MSG2
                        /
                        /RESTART 6 IS A 16 BIT UNSIGNED COMPARE (CMPR).  FLAGS ARE
                        /SET LIKE HL - DE.  STACK USAGE:  2 BYTES.
                        /
            060 000             *60
000 060 174                     LA H
000 061 222                     SU D
000 062 300                     RFZ
000 063 175                     LA L
000 064 223                     SU E
000 065 311                     RET
000 066 377 004                 DSD SPRS        /ADR OF ADR OF ADR OF STACK RESET
                        /
                        /RESTART 7 IS OPEN FOR INTERRUPT USE.
                        /A RETURN IS PUT THERE SO INTERRUPTS WILL BE IGNORED TILL
                        /IT IS PATCHED OUT.  THREE BYTES ARE LEFT FOR A JUMP
                        /TO AN INTERRUPT SERVICE ROUTINE.
                        /
            070 000             *70
000 070 311                     RET
            073 000             *73
                        /
000 073 042 001 005 EXP1,       STHL LRES       /SAVE RESULT
000 076 353                     DE<>HL          /RESTORE TXA
000 077 311                     RET
                        /
000 100 376 040         NXTD,   CPI "           /IGNORE BLANKS
000 102 312 020 000             JTZ NXTC
000 105 376 060                 CPI "0          /<0?
000 107 077                     CMC
000 110 074                     INA             /SET FLAG WTIHOUT AFFECTING CARRY
000 111 075                     DCA
000 112 311                     RET
000 113 302 124 000 TST1,       JFZ NGOT        /NO MATCH
000 116 043                     INX HL          /MATCH - IGONRE IFNOT ADR
000 117 043                     INX HL
000 120 343                     SP<>HL          /RESTORE TXA
000 121 303 020 000             JMP NXTC        /FOUND IT, INCR TXA AND SET FLAGS
000 124 176             NGOT,   LA M            /LOW ORDER IFNOT ADR -> A
000 125 043                     INX HL
000 126 146                     LH M            /IFNOT ADR ON STK, RESTORE TXA
000 127 157                     LL A
000 130 343                     SP<>HL
000 131 311                     RET
                        /
000 132 362 051 000 MSG2,       JFS MSG1        /BIT 7 WAS LOW, PRINT MORE
000 135 343                     SP<>HL          /WAS HIGH, TIME TO RETURN
000 136 311                     RET
                        /
                        /                                               EP 2
                        /ERRO IS THE ERROR MESSAGE PRINTER.  IT MUST ALLWAYS BE
                        /CALLED, THE RETURN ADDRESS IS USED AS THE ERROR NUMBER.
                        /
000 137 315 142 000 SNER,       CAL ERRO        /SYNTAX ERROR TO BE JUMPED TO
000 142 357             ERRO,   MSGP            /PRINT "ERROR"
000 143 015                     DSS 15,12,"E,"R,"R,"O,"R+200
000 144 012
000 145 105
000 146 122
000 147 122
000 150 117
000 151 322
000 152 343                     SP<>HL          /PRINT ERROR ADDRESS
000 153 315 064 004             CAL HLPT
000 156 315 052 004             CAL INPT        /PRINT LINE NUMBER OF ERROR
000 161 052 011 005             LDHL SSTM       /START OF LAST STATEMENT -> HL
000 164 053                     DCX HL
000 165 321                     POPX DE         /ERROR TXA -> DE
000 166 367             ERRP,   CMPR            /AT BAD SPOT YET?
000 167 302 174 000             JFZ ERRQ        /NOPE - PRINT A CHR
000 172 357                     MSGP            /YUP - INSERT A "?"
000 173 277                     DSS "?+200
000 174 327             ERRQ,   FETCH           /END OF STMT?
000 175 312 204 000             JTZ ENTR        /YUP - BACK TO COMMAND MODE
000 200 347                     DEVO            /NOPE - PRINT ONE CHR AND
000 201 303 166 000             JMP ERRP        /KEEP TRYING
                        /
                        /MAIN INTERPRETER ENTRY AND RE-ENTRY POINT.  ENTR SENDS CRLF
                        /AND ENTERS LINE INPUT MODE.  NOCR DOES SAME, WITHOUT CRLF.
                        /NUMBERED LINES ARE EDITED INTO TEXT BUFFER.
                        /UN-NUMBERED LINES ARE PASSED TO STMT FOR EXECUTION.
                        /
000 204 315 154 005 ENTR,       CAL CRLF
000 207 315 024 004 NOCR,       CAL RSSP        /RESET 8080 STACK
000 212 357                     MSGP            /PRINT PROMPT PERIOD "."
000 213 256                     DSS ".+200
000 214 041 377 377             LXI HL '377 377 /SET IMMEDIATE MODE FLAG
000 217 042 262 004             STHL CURL
000 222 315 016 002             CAL GETL        /FETCH AN INPUT LINE
000 225 327                     FETCH           /BLANK LINE?
000 226 312 207 000             JTZ NOCR        /YUP - IGNORE
000 231 322 055 001             JFC STMU        /NOT NUMERIC - EXECUTE IT
```

```
                        /
                        /START OF LINE TEXT EDITOR
                        /
                        /TEXT BUFFER FORMAT:
                        /
                        /       000
                        /BOTX,  LINE 1
                        /       LINE 2
                        /       LINE 3
                        /       000
                        /EOTX,  000
                        /
                        /LINE STORAGE FORMAT:
                        /
                        /       LINE NUMBER LOW 8 BITS
                        /       LINE NUMBER HIGH 8 BITS
                        /       CHRS WHICH APPEAR ON LINE
                        /       000
                        /
000 234 315 205 004             CAL DEINT       /GET LINE # -> DE
000 237 345                     PSHX HL         /FIRST CHR ADR SAVE
000 240 325                     PSHX DE         /SAVE LINE #
000 241 365                     PSHX PSW        /ZERO TRUE IF BLANK LINE
000 242 001 002 000             LXI BC 2        /LINE LENGTH 3 BYTE OVERHEAD
000 245 176             EDT,    LA M            /COUNT UP LINE LENGTH -> BC
000 246 267                     OR A
000 247 043                     INX HL
000 250 003                     INX BC
000 251 302 245 000             JFZ EDT         /KEEP COUNTING
000 254 361                     POPX PSW        /RESTORE FLAGS
000 255 305                     PSHX BC         /SAVE LINE LENTH
000 256 365                     PSHX PSW        /SAVE FLAGS
000 257 315 370 000             CAL LFND        /INSERT ADR -> BC, SONL -> HL
000 262 305                     PSHX BC         /SAVE INSERT ADR
000 263 322 310 000             JFC EDT2        /COULDN'T FIND, SO INSERT ONLY
000 266 353                     DE<>HL          /SONL -> DE
000 267 052 260 004             LDHL EOTX
000 272 032             EDT1,   LDAX DE         /DELETE OLD LINE
000 273 002                     STAX BC
000 274 003                     INX BC
000 275 023                     INX DE
000 276 367                     CMPR            /DONE YET?
000 277 322 272 000             JFC EDT1        /NOPE
000 302 140                     LH B            /SAVE NEW EOTX
000 303 151                     LL C
000 304 053                     DCX HL
000 305 042 260 004             STHL EOTX
000 310 321             EDT2,   POPX DE         /INSERT ADR -> DE
000 311 361                     POPX PSW        /ANYTHING TO INSERT?
000 312 312 207 000             JTZ NOCR        /NOPE - EXIT EDITOR
000 315 052 260 004             LDHL EOTX
000 320 343                     SP<>HL          /EOTX -> BC, LL -> HL
000 321 301                     POPX BC
000 322 011                     DADX BC         /NEW EOTX -> HL
000 323 345                     PSHX HL         /SAVE IT
000 324 315 367 001             CAL EOM?        /ROOM FOR THIS LINE?
000 327 305                     PSHX BC
000 330 343                     SP<>HL          /OEOTX -> HL, NEOTX -> BC
000 331 301                     POPX BC
000 332 367             ED21,   CMPR            /MOVE UP FOR NEW LINE
000 333 176                     LA M            /FROM OEOTX -> NEOTX
000 334 002                     STAX BC
000 335 013                     DCX BC
000 336 053                     DCX HL
000 337 302 332 000             JFZ ED21        /NOT DONE YET
000 342 341                     POPX HL         /RESTORE NEOTX
000 343 042 260 004             STHL EOTX
000 346 353                     DE<>HL          /INSERT ADR -> HL
000 347 321                     POPX DE         /LINE # -> DE
000 350 163                     LM E            /PUT IN NEW LINE #
000 351 043                     INX HL
000 352 162                     LM D
000 353 043                     INX HL
000 354 321                     POPX DE         /ADR OF TEXT ON LINE
000 355 032             EDT3,   LDAX DE         /PUT IT IN BUFFER
000 356 167                     LM A
000 357 043                     INX HL
000 360 023                     INX DE
000 361 267                     OR A
000 362 302 355 000             JFZ EDT3        /NOT DONE INSERTING
000 365 303 207 000             JMP NOCR        /GET ANOTHER LINE
                        /

                        /
                        /LFND IS THE LINE FINDER.
                        /TRIES TO FIND THE LINE # IN DE IN THE BUFFER.
                        /IT WILL EITHER FIND IT, OR HIT THE EOB FIRST, OR GO
                        /ONE LINE PAST BUT NOT HIT EOB.  RETURN CONDITIONS FOLLOW:
                        /
                        /                       IF
                        /                       --
                        /               EOB     GOT IT  NEXT >
                        /               ---     ------  ------
                        /       HL      EOB     SONL    SO>L
                        /       BC      EOB     SOL     SO<L
                        /       CARRY   FALSE   TRUE    FALSE
                        /       ZERO    TRUE    TRUE    FALSE
                        /
                        /USES ALL REGS AND FLAGS EXCEPT DE.  STACK USAGE:  6 BYTES.
                        /
000 370 052 256 004 LFND,       LDHL BOTX       /START AT BEGINNING OF TEXT
000 373 104             LFNE,   LB H            /SAVE START OF LINE -> BC
000 374 115                     LC L
000 375 176                     LA M            /EOB?
000 376 043                     INX HL
000 377 266                     OR M
001 000 053                     DCX HL
001 001 310                     RTZ             /YUP - ZERO TRUE, CARRY FALSE
001 002 176                     LA M            /RELOAD LOW ORDER -> A
001 003 043                     INX HL
001 004 345                     PSHX HL         /SAVE SOL TXA+1
001 005 146                     LH M            /LINE # -> HL
001 006 157                     LL A
001 007 367                     CMPR            /LINE # WE WANT ?
001 010 341                     POPX HL         /SOL+1 -> HL
001 011 365                     PSHX PSW        /SAVE RESULT OF COMPARE
001 012 043                     INX HL          /START OF NEXT LINE -> HL
001 013 315 355 001             CAL FSNL
001 016 361                     POPX PSW        /RESTORE RESULT OF COMPARE
001 017 077                     CMC             /FOUND IT?
001 020 310                     RTZ             /YUP - CARRY, ZERO TRUE
001 021 077                     CMC             /PAST IT?
001 022 320                     RFC             /YUP - CARRY, ZERO FALSE
001 023 303 373 000             JMP LFNE        /NOPE - KEEP LOOKING
                        /
                        /THIS IS THE INTERPRETER CONTROL SECTION.
                        /

                        /STMT IS THE STATEMENT EXECUTOR.  ENTER IT WITH THE TXA
                        /OF THE STRING TO BE EXECUTED -1 IN HL.  KEEPS GOING TILL:
                        /IT FINDS LINE # 0, CONTROL C (^C) ABORT, OR GOTO
                        /-1 (MINUS ONE).  IT PUSHES THE ADDRESS OF RTRN
                        /BEFORE LEAVING, SO WHEN THE STMT HANDLER RETURNS, IT SHOWS
                        /UP AT RTRN.  AT RTRN, TXA SHOULD POINT TO COLON (:) OR
                        /END OF LINE NULL.
                        /
001 026 315 160 005 RTRN,       CAL ABRT        /TEST FOR CONTROL C (^C)
001 031 176                     LA M            /MORE ON THIS LINE?
001 032 376 072         COLN,   CPI ":
001 034 312 054 001             JTZ STMT        /YUP - EXECUTE IT
001 037 267                     OR A            /END OF LINE?
001 040 304 142 000             CFZ ERRO        /NOPE - ILLEGAL TERMINATION CHR
001 043 043                     INX HL          /MOVE TO SONL
001 044 315 100 002             CAL FELN        /LINE # -> DE, RE-ENTER IF EOB
001 047 353                     DE<>HL          /MAKE IT CURRENT LINE
001 050 042 262 004             STHL CURL
001 053 353                     DE<>HL
001 054 327             STMT,   FETCH
001 055 042 011 005 STMU,       STHL SSTM       /SAVE THE START OF THIS STATEMENT
001 060 021 026 001             LXI DE RTRN     /PUSH DESIRED RETURN ADR
001 063 325                     PSHX DE
001 064 310                     RTZ
001 065 317 077                 TST "?          /A PRINT STMT?
001 067 363 005         SIPK,   IFNOT NPRT      /MIGHT BE POKED TO NPRU
001 071 312 154 005 PRT1,       JTZ CRLF
001 074 310             CR?,    RTZ             /RETURN WITH NO CR IF TERMINATOR
001 075 317 073                 TST ";
001 077 104 001                 IFNOT PCOM
001 101 303 074 001             JMP CR?         /IGNORE SEMICOLONS - NO CR IF EOS
001 104 317 054         PCOM,   TST ",          /A COMMA ?
001 106 116 001                 IFNOT QUOT
001 110 076 011                 LAI 11          /YUP - SEND A TAB
001 112 347                     DEVO
001 113 303 074 001             JMP CR?         /NO CR IF EOS
001 116 176             QUOT,   LA M            /LEADING SLASH FOR LITERAL ?
001 117 376 057                 CPI "/
001 121 302 144 001             JFZ PXCL        /NOPE - TRY CHR$
001 124 043                     INX HL          /YUP - MOVE OVER SLASH
001 125 176             QUOS,   LA M            /FETCH A CHR
001 126 267                     OR A            /END OF LINE ?
001 127 314 142 000             CTZ ERRO        /YUP - NO CLOSING SLASH ERROR
001 132 043                     INX HL
001 133 376 057                 CPI "/          /FINAL SLASH?
001 135 312 165 001             JTZ PEXQ        /YUP
001 140 347                     DEVO            /NOPE - SEND IT
001 141 303 125 001             JMP QUOS        /DO MORE
001 144 317 076         PXCL,   TST ">          /A CHR$ FUNCTION ?
001 146 340 005         PXPK,   IFNOT PRI)      /TRY STRING PRINT, MIGHT BE POKED
001 150 337                     EXPR
001 151 173                     LA E            /TRUNCATED EXPR -> A
001 152 347                     DEVO            /SEND IT
001 153 303 165 001             JMP PEXQ
001 156 337             PEXP,   EXPR            /MUST BE AN EXPRESSION
001 157 345                     PSHX HL         /SAVE HL DURING PRINT
001 160 353                     DE<>HL          /NUMBER TO PRINT -> HL
001 161 315 170 004             CAL SHLP        /PRINT THE SIGNED NUMBER
001 164 341                     POPX HL         /RESTORE TXA
001 165 053             PEXQ,   DCX HL          /SET Z FLAG IF EOS
001 166 327                     FETCH
001 167 303 071 001             JMP PRT1
001 172 345             NPRU,   PSHX HL         /SAVE SOL TXA ON STK
001 173 043                     INX HL
001 174 317 075                 TST "=          /SECOND CHR "=" ?
001 176 313 003                 IFNOT CMD?      /NOPE - MUST BE A COMMAND OR ARRAY
001 200 337                     EXPR            /YUP - EVALUATE RIGHT SIDE
001 201 343                     SP<>HL          /SOL -> HL, EOL ON STK
001 202 317 056                 TST ".          /A GOTO STMT ?
001 204 232 001                 IFNOT LEF$
001 206 341             GOT,    POPX HL         /EOS -> HL
001 207 172             GOTA,   LA D            /GOTO LINE ZERO?
001 210 263                     OR E
001 211 310                     RTZ             /YUP - FALL THRU TO NEXT STMT
001 212 172                     LA D            /GOTO LINE 65535 ? (-1)
001 213 243                     ND E
001 214 074                     INA
001 215 312 204 000             JTZ ENTR        /YUP - THIS IS A STOP
001 220 315 370 000 GOTB,       CAL LFND        /FIND HIS LINE
001 223 140                     LH B            /NEW LINE TXA -> HL
001 224 151                     LL C
001 225 053                     DCX HL          /MOVE TO PRECEDING LINES NULL
001 226 330                     RTC             /FOUND THE LINE - EXECUTE IT
001 227 315 142 000             CAL ERRO        /NO FIND ERROR

001 232 317 044         LEF$,   TST "$          /A GOSUB ?
001 234 253 001                 IFNOT MEMA
001 236 315 355 001             CAL FSNL        /FIND START OF NEXT LINE
001 241 176                     LA M            /SAVE ITS LINE NUMBER
001 242 043                     INX HL
001 243 146                     LH M
001 244 157                     LL A
001 245 042 003 005             STHL DLAD
001 250 303 206 001             JMP GOT         /DO A GOTO
001 253 317 041         MEMA,   TST "!          /SET A MEMORY ADDRESS ?
001 255 265 001                 IFNOT POK?
001 257 353                     DE<>HL          /STORE NEW MEMORY ADDRESS
001 260 042 016 005             STHL MADR
001 263 341                     POPX HL         /RESTORE EOS TXA
001 264 311                     RET
001 265 317 046         POK?,   TST "&          /A POKE ?
001 267 277 001                 IFNOT OUT?
001 271 052 016 005             LDHL MADR       /GET THE SET MEMORY ADDRESS
001 274 163                     LM E            /POKE IT WITH LOW ORDER EXPR
001 275 341                     POPX HL         /RESTORE EOS TXA
001 276 311                     RET
001 277 317 137         OUT?,   TST "←          /AN OUT INST ?
001 301 315 001                 IFNOT PAD?      /NOPE
001 303 076 323                 LAI 323         /OUT INST BINARY -> MEMORY (RAM)
001 305 062 013 005             STA RAMIO
001 310 173                     LA E            /DATA TO OUTPUT -> A
001 311 341                     POPX HL         /RESTORE EOS TXA
001 312 303 013 005             JMP RAMIO       /DO THE OUT, AND RETURN
001 315 317 100         PAD?,   TST "@          /SET PORT NUMBER ?
001 317 327 001                 IFNOT DEF?
001 321 173                     LA E            /TRUNCATED EXPRESSION -> A
001 322 062 014 005             STA RAMIO+1     /SET NEW PORT NUMBER INTO RAM
001 325 341                     POPX HL         /RESTORE EOS TXA
001 326 311                     RET
001 327 317 136         DEF?,   TST "↑          /DEFINE A FUNCTION ?
001 331 310 005         DFPK,   IFNOT BSES      /MIGHT BE POKED TO LETS
001 333 043                     INX HL          /MOVE TXA TO EXPRESSION
001 334 042 005 005             STHL DEFF       /SAVE FUNCTIONS TXA
001 337 341                     POPX HL         /RESTORE EOS TXA
001 340 311                     RET
```

```
001 341 315 202 005 LETS,   CAL LOKU        /GET THE INDES OF THE VAR
001 344 163                 LM E            /STORE THE VAL IN MEMORY
001 345 302 352 001         JFZ LETT        /DON'T WRITE HI BYTE IF SINGLE ARRAY
001 350 043                 INX HL
001 351 162                 LM D
001 352 341         LETT,   POPX HL         /IGNORE TXA FROM LOKU
001 353 341                 POPX HL         /RESTORE EOS TXA
001 354 311                 RET
                    /FSNL FINDS THE START OF THE NEXT LINE IN MEMORY.
                    /HL IS BUMPED TO POINT TO THE LO ORDER LINE NUMBER OF THAT
                    /LINE.  A & PSW GET MUNCHED.  STACK USAGE:  2 BYTES.
                    /
001 355 176         FSNL,   LA M
001 356 043                 INX HL
001 357 267                 OR A            /ENDING NULL YET?
001 360 302 355 001         JFZ FSNL
001 363 311                 RET
                    /
                    /EOM AND EOM? CHECK TO MAKE SURE THAT THERE IS AT LEAST 24
                    /BYTES OF STK SPACE LEFT FOR NORMAL OPERATIONS.  EOM? LOOKS
                    /FOR 24 BETWEEN HL AND CURRENT SP.  EOM LOOKS FOR 24 BETWEEN
                    /CURRENT EOTX AND SP.  BOTH MUNCH PSW & A.
                    /
001 364 052 260 004 EOM,    LDHL EOTX       /CURRENT EOTX
001 367 325         EOM?,   PSHX DE         /SAVE DE
001 370 353                 DE<>HL          /SAVE HIS HL
001 371 041 350 377         LXI HL -024     /LOOKING FOR 24 BYTES
001 374 071                 DADX SP         /ADD IN CURRENT SP
001 375 367                 CMPR            /SUBTRACT PASSED HL
001 376 353                 DE<>HL          /RESTORE HIS HL
001 377 321                 POPX DE         /RESOTRE DE
002 000 320                 RFC             /PLENTY OF ROOM LEFT - RETURN
002 001 315 142 000         CAL ERRO        /OUT OF MEMORY ERROR
                    /


                    /
                    /THIS ROUTINE INPUTS A LINE OF TEXT AND PLACES IT
                    /AT LINB WHEN ENTERED AT GETL.  RUBOUT
                    /DELETES THE PREVIOUS CHR.  CONTROL U (^U) DELETES THE
                    /ENTIRE LINE BEING TYPED AND STARTS OVER.  A MAXIMUM
                    /OF LBUL CHRS WILL BE ACCEPTED AFTER WHICH THE
                    /BELL WILL RING INSTEAD OF ECHOING CHRS AS NORMAL.
                    /CONTROL CHRS OTHER THAN CONTROL U, CONTROL G (BELL),
                    /AND CARRIAGE RETURN WILL NOT BE ECHOED BUT IGNORED.
                    /ROUTINE RETURNS ON ENTRY OF A CARRIAGE RETURN BY
                    /ECHOING A CRLF AND PLACING 3 NULLS AT THE END OF BUFFER.
                    /ON EXIT, HL POINT LINB-1.  STACK USAGE:  10 BYTES.
                    /
002 004 053         GETJ,   DCX HL          /DECR CHR POINTER
002 005 357                 MSGP            /SEND A BACK SLASH
002 006 334                 DSS "\+200
002 007 005                 DCB             /DECR CHR COUNTER
002 010 302 023 002         JFZ GETM        /DELETED TOO MANY? - NOPE
002 013 315 154 005 GETK,   CAL CRLF
002 016 041 264 004 GETL,   LXI HL LINB     /CHRS WILL GO HERE
002 021 006 001             LBI 1           /INITIALIZE CHR COUNT
002 023 315 117 005 GETM,   CAL TTYI        /GET CHR -> A
002 026 376 007             CPI 7           /A BELL?
002 030 312 057 002         JTZ GETN        /YUP - PUT IN BUFFER
002 033 376 015             CPI 15          /A CR?
002 035 312 143 005         JTZ CRLE        /YUP - EXIT THRU CRLF
002 040 376 025             CPI 25          /CONTROL U?
002 042 312 013 002         JTZ GETK        /YUP - START OVER
002 045 376 040             CPI "           /< SPACE, CONTROL CHR ?
002 047 332 023 002         JTC GETM        /YUP - IGNORE
002 052 376 177             CPI 177         /RUBOUT?
002 054 312 004 002         JTZ GETJ        /YUP - IGNOR LAST CHR
002 057 117         GETN,   LC A            /SAVE CHR
002 060 170                 LA B            /GET LINE LENGTH -> A
002 061 376 111             CPI LBUL+1      /COMPARE WITH MAXIMUM
002 063 076 007             LAI 7           /GET READY TO RING BELL IF TOO LONG
002 065 322 074 002         JFC GETO        /RING IT
002 070 171                 LA C            /RESTORE CHR
002 071 161                 LM C            /PUT IT IN BUFFER
002 072 043                 INX HL          /INCR BUFFER POINTER
002 073 004                 INB             /INCR CHR COUNTER
002 074 347         GETO,   DEVO            /ECHO CHR
002 075 303 023 002         JMP GETM        /DO SOME MORE
                    /
                    /THIS ROUTINE FETCHES A LINE NUMBER FROM MEMORY -> DE.
                    /IF IT IS LINE 0 (ZERO), THIS MEANS EOB AND IT GOES TO ENTR.
                    /IF NOT 0, JUST RETURN.  MUNCHES DE & A & FLAGS, BUMPS HL.
                    /HL POINTS TO LOW ORDER ON ENTRY, HI ORDER ON EXIT.
                    /STACK USAGE:  2 BYTES.
                    /
002 100 136         FELN,   LE M            /LO ORDER -> E
002 101 043                 INX HL
002 102 126                 LD M            /HI ORDER -> D
002 103 172                 LA D            /IS DE = 0 ?
002 104 263                 OR E
002 105 300                 RFZ             /NOPE - RETURN
002 106 307                 RST             /YUP - BACK TO COMMAND MODE
                    /


                    /
                    /EXPRESSION EVALUATOR.  USES ALL REGISTERS.  RESULT IS LEFT
                    /IN THE DE REGISTER.  WILL PROBABLY RECURSE AT LEAST ONCE.
                    /
                    /               HIERARCHY
                    /
                    /EVALUATED FIRST        ( )
                    /                       *, /
                    /                       +, -
                    /EVALUATED LAST         <, >, =, #
                    /
                    /OPERATORS ON THE SAME LEVEL ARE EVALUATED LEFT TO RIGHT.
                    /
                    /       <EXPR>  ::=     <SUM> I <SUM><<SUM> I <SUM>><SUM>
                    /                       <SUM>=<SUM> I <SUM>#<SUM>
                    /
                    /STACK USAGE:  >= 10 BYTES.  CALLS EOM BEFORE RECURSING.
                    /
002 107 315 206 002 EXPA,   CAL SUM         /GET LEFT SUM
002 112 317 074     EXPS,   TST "<          /FOLLOWED BY "<" ?
002 114 124 002             IFNOT TRY>
002 116 315 163 002         CAL RSUM        /GET RIGHT SUM AND COMPARE
002 121 320                 RFC             /FALSE - DE = 0
002 122 137                 LE A            /TRUE - MAKE DE = 1
002 123 311                 RET
002 124 317 076     TRY>,   TST ">          /GREATER THAN ?
002 126 137 002             IFNOT TRYE
002 130 315 163 002         CAL RSUM        /GET RIGHT SUM
002 133 330                 RTC             /FALSE
002 134 310                 RTZ             /EQUAL IS FALSE
002 135 137                 LE A            /TRUE
002 136 311                 RET
002 137 317 075     TRYE,   TST "=          /EQUAL TO ?
002 141 151 002             IFNOT TRYN
002 143 315 163 002         CAL RSUM        /GET RIGHT SUM
002 146 300                 RFZ             /NOT EQUAL IS FALSE
002 147 137                 LE A
002 150 311         RET1,   RET
002 151 317 043     TRYN,   TST "#          /NOT EQUAL TO  ?
002 153 150 002             IFNOT RETI      /NO RELOPS - RETURN
002 155 315 163 002         CAL RSUM        /GET RIGHT SUM
002 160 310                 RTZ
002 161 137                 LE A
002 162 311                 RET
                    /
                    /RSUM GETS THE RIGHT SUM AFTER A RELOP HAS BEEN FOUND.
                    /ENTER WITH LEFT SUM IN DE.  AFTER FETCHING THE RIGHT SUM,
                    /RIGHT AND LEFT ARE COMPARED WITH A 16 BIT SIGNED COMPARE.
                    /ON EXIT:  FLAGS ARE SET LIKE LEFT - RIGHT,
                    /DE = 0, A = 1, TXA POINTS TO END OF EXPRESSION.
                    /
002 163 325         RSUM,   PSHX DE         /LEFT ON STK
002 164 315 206 002         CAL SUM         /GET RIGHT SUM -> DE
002 167 343                 SP<>HL          /LEFT -> HL, TXA ON STK
002 170 174                 LA H            /COMPARE SIGN OF LEFT AND RIGHT
002 171 252                 XR D
002 172 362 176 002         JFS SAMS        /SAME SIGN - DON'T SWAP
002 175 353                 DE<>HL
002 176 367         SAMS,   CMPR            /DO THE COMPARE
002 177 341                 POPX HL         /RESTORE TXA
002 200 021 000 000         LXI DE 0        /SETUP RESULT OF RELOP
002 203 076 001             LAI 1           /DO A LE A IF TRUE
002 205 311                 RET
                    /
                    /SUM EVALUATOR.
                    /<SUM>  ::=     <TERM> I <SUM> + <TERM> I <SUM> - <TERM>
                    /
                    /THE VALUE OF THE SUM IS LEFT IN DE ON EXIT.
                    /
002 206 315 246 002 SUM,    CAL TERM        /GET LEFT TERM
002 211 317 053     SUMA,   TST "+          /FOLLOWED BY A "+" ?
002 213 230 002             IFNOT SUN
002 215 325                 PSHX DE         /SAVE LEFT HALF
002 216 315 246 002         CAL TERM        /GET RIGHT HALF
002 221 343         SUM1,   SP<>HL          /LEFT -> HL, TXA ON STACK
002 222 031                 DADX DE         /RIGHT + LEFT -> HL
002 223 353                 DE<>HL          /RESULT -> DE
002 224 341                 POPX HL         /RESTORE TXA
002 225 303 211 002         JMP SUMA        /CHECK FOR MORE SUMS
002 230 317 055     SUN,    TST "-          /FOLLOWED BY "-" ?
002 232 150 002             IFNOT RETI      /NOPE - DONE WITH ALL SUMS
002 234 325                 PSHX DE         /SAVE LEFT TERM
002 235 315 246 002         CAL TERM        /GET RIGHT HALF
002 240 315 246 004         CAL COMD        /DE = -RIGHT
002 243 303 221 002         JMP SUM1        /RESULT = -LEFT + RIGHT
                    /


                    /                                               EP 11
                    /TERM EVALUATOR.
                    /<TERM> ::=     <FACT> I <TERM> * <FACT> I <TERM> / <FACT>
                    /
002 246 315 025 003 TERM,   CAL FACT        /GET LEFT FACT
002 251 317 052     TERA,   TST "*          /FOLLOWED BY AN "*" ?
002 253 340 002             IFNOT TERN      /NOPE - TRY DIVISION
002 255 325                 PSHX DE         /SAVE LEFT FACT
002 256 315 025 003         CAL FACT        /GET RIGHT FACT
002 261 343                 SP<>HL          /LEFT -> HL, TXA ON STACK
002 262 345                 PSHX HL
002 263 041 013 005         LXI HL RAMIO    /NUMBER OF BITS
002 266 066 021             LMI #11
002 270 001 000 000         LXI BC 0        /CLEAR PARTIAL PRODUCT
002 273 172         LOOP,   LA D            /16 BIT DE ROTATE RIGHT
002 274 037                 RAR
002 275 127                 LD A
002 276 173                 LA E
002 277 037                 RAR
002 300 137                 LE A
002 301 065                 DCM             /ONE BIT DONE
002 302 312 331 002         JTZ MULS        /ALL BITS DONE
002 305 343                 SP<>HL
002 306 322 316 002         JFC SKIP        /BIT NOT ONE - SKIP ADD
002 311 345                 PSHX HL
002 312 011                 DADX BC
002 313 104                 LB H
002 314 115                 LC L
002 315 341                 POPX HL
002 316 267         SKIP,   OR A            /CLEAR CARRY
002 317 175                 LA L            /16 BIT HL ROTATE LEFT
002 320 027                 RAL
002 321 157                 LL A
002 322 174                 LA H
002 323 027                 RAL
002 324 147                 LH A
002 325 343                 SP<>HL
002 326 303 273 002         JMP LOOP
002 331 321         MULS,   POPX DE         /CLEAN JUNK OFF STACK
002 332 120                 LD B            /RESULT -> DE
002 333 131                 LE C
002 334 341                 POPX HL         /RESTORE TXA
002 335 303 251 002         JMP TERA        /LOOK FOR ADDITIONAL OPERATORS
002 340 317 057     TERN,   TST "/          /FOLLOWED BY "/" ?
002 342 150 002             IFNOT RETI      /NOPE - DONE WITH ALL FACTORS
002 344 325                 PSHX DE         /SAVE LEFT FACT
002 345 315 025 003         CAL FACT        /GET RIGHT FACT
002 350 315 235 004         CAL CHSG        /CHANGE SIGN IF NEEDED
002 353 343                 SP<>HL          /TXA ON STK, LEFT -> HL
002 354 353                 DE<>HL          /LEFT -> DE, ABS(RIGHT) -> HL
002 355 315 237 004         CAL CHS?        /ABS(LEFT) -> DE
002 360 325                 PSHX BC         /SAVE SIGN OF RESULT
002 361 104                 LB H
002 362 115                 LC L            /ABS(RIGHT) -> BC
002 363 353                 DE<>HL          /ABS(LEFT) -> HL
002 364 170         DV02,   LA B            /DIVISION BY ZERO?
002 365 261                 OR C
002 366 314 142 000         CTZ ERRO        /YUP - ERROR
002 371 021 000 000         LXI DE 0        /CLEAR QUOTIENT
002 374 175         DIV1,   LA L            /LEFT = LEFT -RIGHT
002 375 221                 SU C
002 376 157                 LL A
002 377 174                 LA H
003 000 230                 SB B
003 001 147                 LH A
003 002 023                 INX DE          /QUO=QUO + 1
003 003 322 374 002         JFC DIV1        /STILL POSITIVE - SUB AGAIN
003 006 033                 DCX DE          /TOO FAR - QUO = QUO -1
003 007 011                 DADX BC         /GET REMAINDER -> HL
003 010 042 020 005         STHL RMDR       /SAVE IT
003 013 301                 POPX BC         /GET THE SIGN OF RESULT
003 014 170                 LA B
003 015 267                 OR A
```

```
003 016 374 246 004          CTS COMD        /COMPLIMENT RESULT MAYBE
003 021 341                  POPX HL         /RESTORE TXA
003 022 303 251 002          JMP TERA        /LOOK FOR ADDITONAL OPERATORS
                             /

                             /
                             /FACTOR EVALUATOR.
                             /<FACT> ::=     <CONSTANT> I <VARIABLE> I -<FACT>
                             /               +<FACT> I (<EXPR>) I . I ? I $ I %
                             /               ! I & I @ I ↑ I \
                             /
                             /VALUE OF FACTOR LEFT IN DE ON EXIT.
                             /
003 025 317 053      FACT,   TST "+          /UNARY PLUS ?
003 027 031 003              IFNOT FACA      /IGNORE IT
003 031 053          FACA,   DCX HL          /IS THIS A CONSTANT?
003 032 327                  FETCH           /SET FLAGS, TC IS 0 - 9, TZ IS TERMN
003 033 332 205 004          JTC DEINT       /YUP - GET VAL -> DE AND EXIT
003 036 314 142 000          CTZ ERRO        /MISSING EXPRESSION ERROR
003 041 317 055              TST "-          /UNARY MINUS ?
003 043 053 003              IFNOT TRY.
003 045 315 025 003          CAL FACT        /GET FACTOR TO NEGATE
003 050 303 246 004          JMP COMD        /COMPLIMENT IT, RETURN FROM COMD
003 053 317 056      TRY.,   TST "."         /CURRENT LINE ?
003 055 065 003              IFNOT TRY$
003 057 353                  DE<>HL          /SAVE TXA IN DE
003 060 052 262 004          LDHL CURL       /GET CURRENT LINE # -> DE
003 063 353                  DE<>HL          /RESTORE TXA
003 064 311                  RET
003 065 317 044      TRY$,   TST "$          /RETURN ADDRESS ?
003 067 077 003              IFNOT TRY%
003 071 353                  DE<>HL          /SAVE TXA
003 072 052 003 005          LDHL DLAD       /GET RETURN ADDRESS -> HL
003 075 353                  DE<>HL          /RESTORE TXA
003 076 311                  RET
003 077 317 045      TRY%,   TST "%          /DIVISION REMAINDER ?
003 101 111 003              IFNOT TRY!
003 103 353                  DE<>HL          /SAVE TXA
003 104 052 020 005          LDHL RMDR       /GET REMAINDER -> HL
003 107 353                  DE<>HL          /RESTORE TXA
003 110 311                  RET
003 111 317 041      TRY!,   TST "!          /PEEK ?
003 113 126 003              IFNOT TRY&
003 115 353                  DE<>HL          /SAVE TXA
003 116 052 016 005          LDHL MADR       /GET LAST MEMORY ADDRESS -> HL
003 121 176                  LA M            /PEEK -> A
003 122 353                  DE<>HL          /RESTORE TXA
003 123 303 142 003          JMP ARET        /RETURN VALUE IN A REG
003 126 317 046      TRY&,   TST "&          /PORT INPUT ?
003 130 146 003              IFNOT TRY↑
003 132 076 333              LAI 333
003 134 062 013 005          STA RAMIO       /SETUP INP INST IN RAM
003 137 315 013 005          CAL RAMIO       /EXECUTE IT
003 142 137          ARET,   LE A            /SETUP TWO BYTE VALUE -> DE
003 143 026 000              LDI 0
003 145 311                  RET
003 146 317 136      TRY↑,   TST "↑          /USER DEFINED FUNCTION REFERENCE ?
003 150 164 003              IFNOT TRY←
003 152 345                  PSHX HL         /SAVE TXA
003 153 315 364 001          CAL EOM         /VERIFY ROOM FOR RECURSION
003 156 052 005 005          LDHL DEFF       /TXA OF DEFINITION
003 161 337                  EXPR            /EVALUATE THE FUNCTION
003 162 341                  POPX HL         /RESTORE TXA
003 163 311                  RET
003 164 317 137      TRY←,   TST "←          /RESULT OF LAST EXPRESSION ?
003 166 176 003              IFNOT USR?
003 170 353                  DE<>HL          /SAVE TXA
003 171 052 001 005          LDHL LRES       /GET LAST EXPR RESULT -> HL
003 174 353                  DE<>HL
003 175 311                  RET
003 176 317 100      USR?,   TST "@          /MACHINE LANGUAGE CALL ?
003 200 220 003              IFNOT TRY\
003 202 345                  PSHX HL         /SAVE TXA
003 203 315 364 001          CAL EOM         /ENUF STACK SPACE ?
003 206 052 007 005          LDHL USRL       /HIS ROT ADR ON STK, TXA -> HL
003 211 343                  SP<>HL
003 212 311                  RET             /GOTO TO HIS ROT
                             /
                             /SAMPLE USR ROT TO RETURN THE ASCII VALUE OF THE CHR
                             /FOLLOWING THE @.
                             /
003 213 136          USR,    LE M            /SETUP TWO BYTE VALUE -> DE
003 214 026 000              LDI 0
003 216 043                  INX HL          /MOVE TXA OVER CHR
003 217 311                  RET
                             /
003 220 317 134      TRY\,   TST "\          /SINGLE CHR INPUT ?
003 222 232 003              IFNOT TRY(
003 224 315 117 005          CAL TTYI        /GET THE INPUT -> A
003 227 303 142 003          JMP ARET
003 232 317 050      TRY(,   TST "(          /EXPRESSION IN PARENTHESIS ?
003 234 266 003              IFNOT TRY?
003 236 345                  PSHX HL         /MAKE SURE THERE IS ROOM BEFORE
003 237 315 364 001          CAL EOM         /RECURSING
003 242 341                  POPX HL
003 243 337                  EXPR            /RECURSIVE
003 244 317 051              TST ")          /GOT TO HAVE A RIGHT TO MATCH
003 246 137 000              IFNOT SNER      /NOPE - ERROR
003 250 311                  RET
003 251 315 202 005 TRY),    CAL LOKU        /GET THE VARIABLES INDES -> HL
003 254 136                  LE M            /VAR VAL -> DE
003 255 026 000              LDI 0           /CLEAR HIGH BITS IF SINGLE ARRAY
003 257 302 264 003          JFZ TRYW        /SINGLE BYTE ARRAY, DON'T LOAD HI
003 262 043                  INX HL
003 263 126                  LD M
003 264 341          TRYW,   POPX HL         /RESTORE TXA, PUSHED BY LOKU
003 265 311                  RET
                             /
                             /TRY? WILL HANDLE THE INPUT OPERATOR IF PRESENT.  EXECUTION
                             /WILL STOP AND A "?" WILL BE PRINTED ON THE OUTPUT DEVICE.
                             /THE USER RESPONDS WITH ANY VALID EXPRESSION, AND HITS
                             /RETURN.  IT IS NOT A GOOD IDEA TO TYPE QUESTION MARKS
                             /IN RESPONSE TO AN INPUT STMT. !!!!   MUNCHES LINB.
                             /
003 266 317 077      TRY?,   TST "?          /THE LINE INPUT OPERATOR ?
003 270 251 003              IFNOT TRYV
003 272 357                  MSGP            /SEND THE QUESTION MARK
003 273 077                  DSS "?," +200
003 274 240
003 275 345                  PSHX HL         /SAVE THE TXA
003 276 315 364 001          CAL EOM         /VERIFY ROOM FOR RECURSION
003 301 315 016 002          CAL GETL        /GET HIS INPUT
003 304 327                  FETCH           /GET FIRST CHR, RETURN ?
003 305 312 204 000          JTZ ENTR        /YUP - CLEAR STK AND RE-ENTER
003 310 337                  EXPR            /EVALUATE HIS INPUT   RECURSIVE
003 311 341                  POPX HL         /RESTORE TXA
003 312 311                  RET
```

```
                             /
                             /
                             /COMMAND PROCESSOR.
                             /
003 313 317 133      CMD?,   TST "[          /ARRAY LET STMT ?
003 315 334 003              IFNOT CMD       /NOPE - IT'S A COMMAND
003 317 176          LOP,    LA M            /MOVE TO RIGHT EXPR
003 320 043                  INX HL
003 321 376 135              CPI "]
003 323 302 317 003          JFZ LOP
003 326 043                  INX HL          /MOVE OVER "="
003 327 337                  EXPR            /EXPR VALUE -> DE
003 330 343                  SP<>HL          /EOS TXA ON STK, SOS TXA -> HL
003 331 303 341 001          JMP LETS        /DO THE ASSIGNMENT
003 334 341          CMD,    POPX HL         /RESTORE SOL TXA
003 335 317 114              TST "L          /LIST COMMAND ?
003 337 004 004              IFNOT NEW?
003 341 315 205 004          CAL DEINT       /GET ARG -> DE, 0 IF NO ARG
003 344 315 370 000          CAL LFND        /FIND THAT LINE
003 347 140                  LH B            /START ADDRESS -> HL
003 350 151                  LL C
003 351 076                  DSS 76          /SETUP BOGUS LAI
003 352 043          LISC,   INX HL          /SKIPPED FIRST TIME THRU, FROM LAI
003 353 315 160 005 LISA,    CAL ABRT        /CONTROL C (↑C) CHECK
003 356 315 154 005          CAL CRLF
003 361 315 100 002          CAL FELN        /FETCH LINE # -> DE, EXIT IF ZERO
003 364 345                  PSHX HL         /SAVE DURING PRINT
003 365 353                  DE<>HL          /LINE # -> HL
003 366 315 066 004          CAL NOSP        /PRINT IT
003 371 341                  POPX HL         /FIRST CHR OF LINE
003 372 043          LISB,   INX HL          /GET A CHR
003 373 176                  LA M
003 374 267                  OR A            /EOL?
003 375 312 352 003          JTZ LISC        /LAST ON LINE - DO NEXT LINE
004 000 347                  DEVO            /NOT LAST - PRINT IT
004 001 303 372 003          JMP LISB        /DO REST OF LINE
                             /
004 004 317 116      NEW?,   TST "N          /NEW COMMAND ?
004 006 037 004              IFNOT RUN?
004 010 052 256 004          LDHL BOTX       /PUT EOB MARK IN BUFFER
004 013 257          NEW1,   XR A            /A=0
004 014 167                  LM A
004 015 043                  INX HL
004 016 167                  LM A
004 017 043                  INX HL
004 020 167                  LM A
004 021 042 260 004          STHL EOTX
004 024 301          RSSP,   POPX BC         /RETURN ADDRESS -> BC
004 025 052 377 004          LDHL SPRS       /HOLDS STACK RESET ADDRESS
004 030 371                  SP<HL
004 031 305                  PSHX BC         /RESTORE RETURN ADDESS
004 032 052 256 004          LDHL BOTX       /INCASE THIS IS SUICIDAL
004 035 053                  DCX HL          /BOTX - 1 -> HL
004 036 311                  RET
                             /
004 037 317 122      RUN?,   TST "R          /RUN COMMAND ?
004 041 173 005              IFNOT OS?
004 043 312 024 004          JTZ RSSP        /NO ARG - RESET STACK AND GO
004 046 337                  EXPR            /GET THE ARGUMENT
004 047 303 207 001          JMP GOTA        /DO A GOTO
                             /
                             /THESE ROUTINES ARE USED TO PRINT THE 16 BITS IN THE
                             /HL REGISTER AS DECIMAL ASCII ON THE TERMINAL.  INPT PRINTS
                             /THE NUMBER IN CURL IF IT IS NOT 65535 (NOT IMMEDIATE MODE).
                             /THE WORD "IN" PRECEDES THE NUMBER IF IT IS PRINTED.  SHLP
                             /PRINTS A 15 BIT SIGNED NUMBER IN HL (-32768 TO 32767).
                             /HLPT PRINTS THE 16 BIT UNSIGNED NUMBER IN HL (0 TO 65535).
                             /NOSP PRINTS 16 BIT UNSIGNED NUMBERS IN HL WITHOUT THE
                             /LEADING SPACE NORMALLY PRINTED.  ALL NUMBERS ARE FOLLOWED
                             /BY ONE TRAILING SPACE.  SHLP PRINTS A MINUS SIGN ("-")
                             /IN PLACE OF THE LEADING SPACE IF HL IS NEGATIVE.
                             /STACK USAGE:  8 BYTES.  MUNCHES ALL REGS.
                             /
004 052 052 262 004 INPT,    LDHL CURL       /CURRENT LINE NUMBER -> HL
004 055 174                  LA H            /IS IT 377 377
004 056 245                  ND L
004 057 074                  INA
004 060 310                  RTZ             /YUP - RETURN PRINT NOTHING
004 061 357                  MSGP            /NOPE - PRINT "IN"
004 062 111                  DSS "I,"N+200
004 063 316
004 064 357          HLPT,   MSGP            /PRINT A SPACE
004 065 240                  DSS " +200
004 066 021 156 004 NOSP,    LXI DE TENS     /POINT TO POWERS OF TEN TABLE
004 071 325                  PSHX DE         /PUT TABLE ADR ON STACK
004 072 016 001              LCI 1           /CLEAR SIGNIFICANT DIGIT FLAG
004 074 343          POSI,   SP<>HL          /NUMBER ON STK, TABLE -> HL
004 075 136                  LE M            /POWER OF TEN -> DE
004 076 043                  INX HL
004 077 126                  LD M
004 100 043                  INX HL
004 101 343                  SP<>HL          /TABLE ON STK, NUMBER -> HL
004 102 006 000              LBI 0           /THIS DIGIT = 0
004 104 175          DIVD,   LA L            /16 BIT SUBTRACT HL = HL - DE
004 105 223                  SU E
004 106 157                  LL A
004 107 174                  LA H
004 110 232                  SB D
004 111 147                  LH A
004 112 004                  INB             /INCREMENT THIS DIGIT
004 113 322 104 004          JFC DIVD        /NOT NEGATIVE YET - KEEP SUBTRACING
004 116 005                  DCB             /GONE ONE TOO FAR, DIGIT = DIGIT -1
004 117 031                  DADX DE         /GONE TOO FAR, ADD BACK TEN POWER
004 120 257                  XR A            /A=0
004 121 260                  OR B            /IS THIS DIGIT ZERO ?
004 122 302 131 004          JFZ PRNT        /NOPE - PRINT IT
004 125 261                  OR C            /ANY SIGNIFICANT DIGITS YET ?
004 126 302 136 004          JFZ BYPA        /NOPE - DON'T PRINT THIS ZERO
004 131 306 060      PRNT,   ADI "0          /ADD IN ASCII BIAS
004 133 016 000              LCI 0           /SET SIGINIFICANT DIGIT FLAG
004 135 347                  DEVO            /SEND THIS DIGIT
004 136 173          BYPA,   LA E            /ON THE LAST DIGIT ?
004 137 075                  DCA
004 140 302 074 004          JFZ POSI        /NOPE - DO NEXT ONE
004 143 321                  POPX DE         /YUP - CLEAN UP STACK
004 144 171                  LA C            /SIGNIFICANT DIGIT FLAG -> A
004 145 267                  OR A            /HAVE WE SENT ANY SIG DIGS YET ?
004 146 312 153 004          JTZ SPOU        /YUP - OUTPUT THE TRAILING SPACE
004 151 357                  MSGP            /NOPE - WE'RE PRINTING A ZERO
004 152 260                  DSS "0+200      /SEND A "0"
004 153 357          SPOU,   MSGP            /PRINT A SPACE
004 154 240                  DSS " +200
004 155 311                  RET
004 156 020 047      TENS,   DSD 010000, 01000, 0100, 010, 01
004 160 350 003
```

```
004 162 144 000
004 164 012 000
004 166 001 000
004 170 353             SHLP,   DE<>HL          /NUMBER -> DE
004 171 315 235 004             CAL CHSG        /ABS(NUMBER) -> DE
004 174 353                     DE<>HL          /ABS(NUMBER) -> HL
004 175 362 064 004             JFS HLPT        /WAS POSITIVE, PRINT SPACE
004 200 357                     MSGP            /PRINT THE MINUS SIGN ("-")
004 201 255                     DSS "-+200
004 202 303 066 004             JMP NOSP        /PRINT THE NUMBER
                        /
                        /
                        /DEINT TAKES ASCII FROM MEMORY INTO BINARY IN DE.
                        /MOVES TXA UNTIL CHR IS NOT 0 - 9.
                        /STACK USAGE:  4 BYTES.  MUNCHES ALL REGS EXCEPT BC.
                        /
004 205 053             DEINT,  DCX HL          /DECR FOR FETCH
004 206 021 000 000             LXI DE 0        /CLEAR PARTIAL SUM
004 211 327             DEIM,   FETCH           /FETCH CHR    0 - 9?
004 212 320                     RFC             /NOPE - DONE
004 213 345                     PSHX HL         /SAVE CHR ADR
004 214 142                     LH D            /PARTIAL SUM -> HL
004 215 153                     LL E
004 216 031                     DADX DE         /HL = DE * 10
004 217 051                     DADX HL         /PS = PS * 10
004 220 031                     DADX DE
004 221 051                     DADX HL
004 222 326 060                 SUI "0          /REMOVE ASCII BIAS
004 224 137                     LE A            /SETUP 16 BIT DIGIT -> DE
004 225 026 000                 LDI 0
004 227 031                     DADX DE         /ADD IN NEW DIGIT
004 230 353                     DE<>HL          /PARTIAL SUM -> DE
004 231 341                     POPX HL         /RESOTRE TXA
004 232 303 211 004             JMP DEIM
                        /
                        /CHS? CHECKS THE SIGN OF DE REG.  IF POSITIVE, RETURN A
                        /MUNCHED, SIGN BIT FALSE.  IF NEGATIVE, COMPLIMENT DE,
                        /A MUNCHED, SIGN BIT SAME AS THAT OF B REG.  CHSG
                        /CLEARS THE SIGN BIT OF B REG FIRST.  COMD UNCONDITIONALLY
                        /COMPLIMENTS DE REG.  STACK USAGE:  2 BYTES.
                        /
004 235 006 000         CHSG,   LBI 0           /CLEAR RESULT SIGN
004 237 172             CHS?,   LA D            /IS DE POSITIVE ?
004 240 267                     OR A
004 241 360                     RFS             /YUP - RETURN
004 242 170                     LA B            /NOPE - FLIP SIGN OF B
004 243 356 200                 XRI 200
004 245 107                     LB A            /AND FALL THRU TO COMPLIMENT DE
004 246 172             COMD,   LA D
004 247 057                     CMA
004 250 127                     LD A
004 251 173                     LA E
004 252 057                     CMA
004 253 137                     LE A
004 254 023                     INX DE
004 255 311                     RET
                        /
                        /
                        /RAM DEFINITIONS
                        /
004 256 154 006         BOTX,   DSD EOP         /ADR OF FIRST CHR IN BUFFER
004 260 156 006         EOTX,   DSD EOP+2       /ADR OF LAST CHR IN BUFFER
004 262 377 377         CURL,   DSD '377 377    /CURRENT LINE NUMBER
        110 000         LBUL=072                /INPUT LINE BUFFER LENGTH
        000 374         OS='374                 /ADDRESS OF OPERATING SYSTEM
004 264 000             LINB,   DSS 0           /LEAVE SPACE FOR INPUT LINE BUF
        377 004                 *.+LBUL+2
004 377 000 010         SPRS,   DSD '10         /ADDRESS OF STACK POINTER RESET
005 001 000 000         LRES,   DSD 0           /HOLDS RESULT OF LAST EXPR EVAL
005 003 000 000         DLAD,   DSD 0           /HOLDS RETURN LINE NUMBER FOR "$"
005 005 033 001         DEFF,   DSD COLN+1      /INITIALIZE TXA OF USER DEFINED FUN
005 007 213 003         USRL,   DSD USR         /ADR OF USERS MACHINE LANG CALL
005 011 154 036         SSTM,   DSD EOP         /INITIALIZE START OF LAST STMT
005 013 323 010         RAMIO,  OUT 10          /RAM AREA FOR INP AND OUT
005 015 311                     RET
005 016 000 000         MADR,   DSD 0           /SAVE AREA FOR PEEK / POKE ADDRESSES
005 020 000 000         RMDR,   DSD 0           /SAVE AREA FOR DIVISION REMAINDER
005 022 000             VART,   DSS 0           /LEAVE ROOM FOR PROGRAM VARIABLES
        106 005                 *.+051
                        /
                        /
                        /
                        /DEVO STARTS AT RESTART 4.
                        /DEVO,  PSHX PSW
                        /       INP 1
                        /       NDI 2
005 106 312 041 000 DEVP,       JTZ DEVQ        /NOT READY
005 111 361                     POPX PSW
005 112 323 010         TODP,   OUT 10
005 114 311                     RET
005 115 000 000                 DSD 0           /PATCH ROOM
                        /
                        /TTYI GETS A CHR FROM THE INPUT DEVICE.  CAN MUNCH A AND
                        /FLAGS.  STACK USAGE:  4 BYTES.
                        /
005 117 315 134 005 TTY1,       CAL TRDY        /IS INPUT READY ?
005 122 312 117 005 TIA,        JTZ TTYI        /NOPE - KEEP TRYING
005 125 333 000         TIDP,   INP 0
005 127 346 177                 NDI 177
005 131 311                     RET
005 132 000 000                 DSD 0           /LEAVE ROOM FOR PATCHES
                        /
                        /TEST TERMINAL INPUT READY BIT STATUS.  MUNCHES A & FLAGS.
                        /STACK USAGE:  2 BYTES.
                        /
005 134 333 001         TRDY,   INP 1           /GET INPUT STATUS
005 136 346 001         TIRM,   NDI 1           /MASK TO INPUT READY BIT
005 140 311                     RET             /FZ MEANS READY, TZ MEANS NOT READY
005 141 000 000                 DSD 0           /ROOM FOR PATCHES
                        /
005 143 257             CRLE,   XR A            /A CONTINUATION OF GETL
005 144 167                     LM A            /PUTS EOB/EOL MARK IN LINB
005 145 043                     INX HL
005 146 167                     LM A
005 147 043                     INX HL
005 150 167                     LM A
005 151 041 263 004             LXI HL LINB-1
                        /
                        /CRLF SEND A CARRIAGE RETURN AND LINE FEED TO TERMINAL.
                        /MUNCHES A & FLAGS.  STACK USAGE:  8 BYTES.
                        /
005 154 357             CRLF,   MSGP
005 155 015                     DSS 15,212
005 156 212
005 157 311                     RET
                        /
                        /ABRT CHECKS THE CONSOLE DEVICE FOR A CHR AND IF THERE,
                        /CHECK IF IT'S A CONTROL C.  IF NOT, RETURN WITH A MUNCHED.
                        /IF YES, FALL THROUGH TO STOP ROUTINE.
                        /STACK USAGE:  6 BYTES.
                        /
005 160 315 134 005 ABRT,       CAL TRDY        /IS DATA READY FLAG UP ?
005 163 310             TIB,    RTZ             /NOPE - RETURN
005 164 315 117 005             CAL TTYI        /YUP - FETCH THE CHR
005 167 376 003                 CPI 3           /A CONTROL C (+C) ?
005 171 300                     RFZ             /NOPE - RETURN
005 172 307                     RST             /BACK TO ENTRY POINT
                        /
                        /
005 173 317 117         OS?,    TST "O          /JMP TO OS ?
005 175 002 006         OSPK,   IFNOT SAV?      /MIGHT BE POKED TO SNER
005 177 303 000 374             JMP OS
                        /
                        /LOKU GETS THE INDES OF THE VARIBLE POINTED TO BY HL AND
                        /RETUNS THEM IN HL.  THE TXA IS BUMPED OVER THE VARNAM,
                        /AND PUSHED BEFORE RETURNING.  YOU MUST POP AFTER CALLING
                        /LOKU.  USES ALL REGS EXCEPT DE.  STACK USAGE.  4 BYTES IF
                        /VAR IS A - Z, >= 16 IF SUBSCRIPTED.  ZERO FLAG IS TRUE
                        /IF IT IS DOUBLE BYTE VAR (SIMPLE OR DOUBLE ARRAY).  ZERO
                        /IS FALSE IF SINGLE BYTE VARIABLE (SINGLE BYTE ARRAY).
                        /
005 202 176             LOKU,   LA M            /VARNAM -> A
005 203 326 101                 SUI "A          /IS IT A - Z ?
005 205 332 232 005 LKP1,       JTC DARY        /NOPE
005 210 376 033                 CPI 027         /26 LETTERS + 1
005 212 322 232 005 LKP2,       JFC DARY        /NOPE - TRY ARRAYS IF NOT POKED
                        /SAVE NEW TXA ON STACK BEFORE RETURNING
005 215 043                     INX HL          /MOVE TXA OVER VARNAM
005 216 343                     SP<>HL          /PUT TXA ON STK
005 217 345                     PSHX HL         /PUT RETURN ADDRESS BACK
005 220 041 022 005             LXI HL VART     /BASE ADDRESS -> HL
005 223 007                     RLC             /MULTIPLY INDEX BY 2
005 224 117                     LC A            /TWO BYTE INDEX -> BC
005 225 006 000                 LBI 0
005 227 011                     DADX BC         /ADD IN INDEX TO BASE
005 230 257                     XR A            /SET ZERO FLAG, THIS IS DOUBLE BYTE
005 231 311                     RET
                        /
        232 005         EOP1=.                  /THIS WILL BE BOTX-1 IF ARRAYS,
                                                /STRING, AND TAPE / SAVE ARE DELETED
                        /
005 232 317 042         DARY,   TST ""          /DOUBLE BYTE ARRAY ?
005 234 252 005                 IFNOT SARY
005 236 315 272 005             CAL SUBS        /GET THE SUBSCRIPT -> BC
005 241 343                     SP<>HL          /INDES -> HL, TXA ON STK
005 242 345                     PSHX HL
005 243 052 336 005             LDHL DBSE
005 246 011                     DADX BC
005 247 011                     DADX BC
005 250 257                     XR A            /SET ZERO FLAG, THIS IS DOUBLE BYTE
005 251 311                     RET
005 252 317 047         SARY,   TST "'          /SINGLE BYTE ARRAY ?
005 254 137 000                 IFNOT SNER
005 256 315 272 005 SSUB,       CAL SUBS        /SUBSCRIPT -> BC
005 261 343                     SP<>HL          /TXA ON STK, INDES -> HL
005 262 345                     PSHX HL
005 263 052 334 005             LDHL SBSE
005 266 011                     DADX BC
005 267 366 001                 ORI 1           /RESET ZERO FLAG, TO SAY SINGLE BYTE
005 271 311                     RET
                        /
                        /SUBS GETS THE SUBSCRIPT FOR A STRING OR ARRAY -> BC.
                        /MUNCHES ALL REGS EXCEPT DE.  STACK USAGE:  >= 14 BYTES.
                        /
005 272 325             SUBS,   PSHX DE         /SAVE DE
005 273 317 133                 TST "[          /IGNORE "["
005 275 277 005                 IFNOT SUB0
005 277 337             SUB0,   EXPR            /GET THE SUBSCRIPT -> DE
005 300 102                     LB D            /SUBSCRIPT -> BC
005 301 113                     LC E
005 302 321                     POPX DE         /RESTORE DE
005 303 317 135                 TST "]          /IGNORE "]"
005 305 307 005                 IFNOT SUB1
005 307 311             SUB1,   RET
                        /
005 310 317 047         BSES,   TST "'          /SET SINGLE BYTE ARRAY BASE ?
005 312 322 005                 IFNOT BSED
005 314 353                     DE<>HL          /NEW BASE -> HL
005 315 042 334 005             STHL SBSE       /SAVE NEW BASE
005 320 341                     POPX HL         /RESTORE EOS TXA
005 321 311                     RET
005 322 317 042         BSED,   TST ""          /SET DOUBLE BYTE ARRAY BASE ?
005 324 341 001                 IFNOT LETS      /MUST BE A LET
005 326 353                     DE<>HL          /NEW BASE -> HL
005 327 042 336 005             STHL DBSE       /SAVE NEW BASE
005 332 341                     POPX HL         /RESTORE EOS TXA
005 333 311                     RET
                        /
005 334 010 370         SBSE,   DSD '370 10     /ADR OF SINGLE BYTE ARRAY BASE ADR
005 336 010 370         DBSE,   DSD '370 10     /ADR OF DOUBLE BYTE ARRAY BASE ADR
                        /
        340 005         EOP2=.                  /THIS WILL BE BOTX-1 IF STRINGS AND
                                                /TAPE / SAVE ARE DELETED.
                        /
005 340 317 051         PRI),   TST ")          /PRINT STRING ARRAY ?
005 342 156 001                 IFNOT PEXP
005 344 315 256 005             CAL SSUB        /GET STRING TXA -> HL, TXA ON STK
005 347 353                     DE<>HL          /STRING TXA -> DE
005 350 341                     POPX HL         /TXA BACK -> HL
005 351 032             STRA,   LDAX DE         /GET A STRING CHR
005 352 267                     OR A            /EOS YET ?
005 353 312 165 001             JTZ PEXQ        /YUP - DO MORE OF ? STMT
005 356 347                     DEVO            /NOPE - PRINT IT
005 357 023                     INX DE          /BUMP STRING TXA
005 360 303 351 005             JMP STRA        /PRINT SOME MORE
                        /
005 363 317 051         NPRT,   TST ")          /STRING INPUT ?
005 365 172 001                 IFNOT NPRU
005 367 315 256 005             CAL SSUB        /GET STRING DESTINATION TXA -> HL
005 372 357                     MSGP            /PRINT PROMPT "- "
005 373 055                     DSS "-," +200
005 374 240
005 375 315 021 002             CAL GETL+3      /USE GETL TO INPUT STRING
006 000 341                     POPX HL         /GET TXA BACK, (PUSHED BY SSUB)
006 001 311                     RET
                        /
        002 006         EOP3=.                  /THIS WILL BOTX-1 IF SAVE / TAPE
                                                /IS DELETED
                        /
```

```
                        /SAV? PUNCHES TAPES OF THE CONTENTS OF THE TEXT
                        /BUFFER.  RETURNS TO COMMAND MODE WHEN DONE.
                        /COMMAND IS FOLLOWED BY A SINGLE CHR PROGRAM NAME SO
                        /MORE THAN ONE PGM CAN BE PUT ON A TAPE.  IF CR IS
                        /GIVEN FOR NAME, PUNCH NAME AS A NULL.
                        /
                        /TAPE FORMAT:
                        /
                        /       252     START CHR
                        /       XXX     NAME OF PROGRAM, 000 IF NULL NAME
                        /       NNN     DATA BYTES BETWEEN BOTX AND EOTX
                        /       000
                        /       000
                        /       000     EOT IS MARKED BY THREE NULLS
                        /
006 002 317 123         SAV?,   TST "S          /SAVE COMMAND ?
006 004 040 006                 IFNOT TAP?
006 006 076 252                 LAI 252         /START OF TAPE CHR
006 010 315 126 006             CAL PNOU        /SEND IT
006 013 176                     LA M            /PROGRAM NAME CHR -> A
006 014 315 126 006             CAL PNOU        /SEND IT
006 017 052 260 004             LDHL EOTX       /STOP ADDRESS -> DE
006 022 353                     DE<>HL
006 023 052 256 004             LDHL BOTX       /START OF TEXT ADR -> HL
006 026 176             SAVA,   LA M            /CHR OF PROGRAM -> A
006 027 315 126 006             CAL PNOU        /SEND IT
006 032 367                     CMPR            /DONE YET ?
006 033 043                     INX HL          /BUMP TXA
006 034 302 026 006             JFZ SAVA        /NOPE - KEEP SAVING
006 037 307                     RST             /ALL DONE, RE-ENTER
                        /
                        /TAP? READS A TAPE FROM THE READER INTO THE TEXT
                        /BUFFER.  RETURNS TO COMMAND MODE WHEN DONE.  COMMAND IS
                        /FOLLOWED BY A SINGLE CHR PROGRAM NAME, LIKE SAVE.
                        /IT WILL SEARCH THE TAPE FOR A START CHR FOLLOWED BY THE
                        /NAME GIVEN.  IF CR IS GIVEN FOR A NAME, TAKE FIRST ONE
                        /FOUND.  IF THE NAMED PROGRAM CAN'T BE FOUND, THE TEXT
                        /BUFFER IS LEFT ALONE.  WHEN READING STARTS, THE NAME
                        /BYTE FROM TAPE IS ECHOED SO YOU'LL KNOW IT IS LOADING.
                        /
006 040 317 124         TAP?,   TST "T          /READ A TAPE COMMAND ?
006 042 137 000                 IFNOT SNER
006 044 315 142 006     TAPA,   CAL CHIN        /GET A CHR
006 047 376 252                 CPI 252         /START CHR ?
006 051 302 044 006             JFZ TAPA        /NOPE - KEEP LOOKING
006 054 315 142 006             CAL CHIN        /YUP - GET NAME CHR
006 057 276                     CP M            /THE ONE WE WANT ?
006 060 312 072 006             JTZ TAPF        /YUP - START READING
006 063 107                     LB A            /SAVE NAME IN B
006 064 176                     LA M            /DID HE GIVE DON'T CARE NAME ?
006 065 267                     OR A
006 066 170                     LA B            /NAME FROM TAPE -> A
006 067 302 044 006             JFZ TAPA        /NOPE - DON'T READ THIS ONE IN
006 072 347             TAPF,   DEVO            /SEND NAME OF PGM BEING READ
006 073 052 256 004             LDHL BOTX       /WHERE IT WILL GO
006 076 016 003         TAPB,   LCI 3           /INITIALIZE EOT NULL COUNTER
006 100 315 142 006     TAPC,   CAL CHIN        /GET A CHR
006 103 167                     LM A            /PUT IN RAM
006 104 315 367 001             CAL EOM?        /PGM TOO BIG ?
006 107 176                     LA M            /GET CHR BACK
006 110 043                     INX HL          /BUMP
006 111 267                     OR A            /A NULL ?
006 112 302 076 006             JFZ TAPB        /NOPE - KEEP READING
006 115 015                     DCC             /DECR EOT NULL COUNT
006 116 302 100 006             JFZ TAPC        /NOT THIRD ONE - KEEP READING
006 121 053                     DCX HL          /STORE NEW EOTX
006 122 042 260 004             STHL EOTX
006 125 307                     RST             /BACK TO COMMAND MODE
                        /

                        /PNOU IS THE PUNCH DRIVER USED BY SAVE.  ENTER WITH CHR TO
                        /SEND IN A REG.  STACK USAGE:  2 BYTES.
                        /
006 126 365             PNOU,   PSHX PSW        /SAVE CHR TO SEND
006 127 333 005         PNOV,   INP 5           /GET PUNCH STATUS
006 131 346 002         CORM,   NDI 2           /READY YET ?
006 133 312 127 006     COA,    JTZ PNOV
006 136 361                     POPX PSW        /IT'S READY, SEND THE CHR
006 137 323 016         CODP,   OUT 16
006 141 311                     RET
                        /
                        /CHIN IS THE READER INPUT ROUTINE CALLED BY THE SAVE
                        /COMMAND.  IT MUNCHES A & FLAGS.  STACK USAGE:  2 BYTES.
                        /
006 142 333 005         CHIN,   INP 5           /GET READER STATUS
006 144 346 001         CIRM,   NDI 1           /READY YET ?
006 146 312 142 006     CIA,    JTZ CHIN        /NOPE - WAIT FOR IT
006 151 333 004         CIDP,   INP 4           /GOT A READY, GET THE INPUT
006 153 311                     RET
                        /
        000 007                 PGE='7          /PAGE FOR BINARY LOADER
        154 006         EOP=.           /THIS IS BOTX-1 IF TAPE / SAVE ARE KEPT
                        /
                        /
                        /INIT IS THE INITIALIZATION ROUTINE.  IT IS LOCATED IN THE
                        /MIDDLE OF THE CASUAL PROGRAM STORAGE AREA.  IT IS ENTERED
                        /WHEN CASUAL IS EXECUTED AFTER LOADING.  IT POKES OUT
                        /THE JUMP TO IT.  RESPOND TO "MEM SIZ ?" WITH THE
                        /DECIMAL NUMBER OF THE HIGHEST ADDRESS TO BE USED BY CASUAL
                        /OR HIT CARRIAGE RETURN TO USE ALL RAM AVAILABLE.
                        /
006 154 061 000 010 INIT,   LXI SP PGE +'1  /SETUP TEMPORARY STACK POINTER
006 157 357                 MSGP            /SEND "MEM SIZ? " MESSAGE
006 160 015                 DSS 15,12,"M,"E,"M," ,"S,"I,"Z,"?+200
006 161 012
006 162 115
006 163 105
006 164 115
006 165 040
006 166 123
006 167 111
006 170 132
006 171 277
006 172 315 016 002         CAL GETL        /GET HIS RESPONSE
006 175 327                 FETCH           /FETCH FIRST CHR, A RETURN ?
006 176 302 221 006         JFZ NUM         /NOPE - GET A NUMBER
006 201 041 247 006         LXI HL MMEM     /START OF RAM SEARCH
006 204 176         INIS,   LA M            /GET A CHR FROM MEMORY
006 205 057                 CMA
006 206 167                 LM A            /WRITE IT BACK COMPLIMENTED
006 207 276                 CP M            /DID IT GO ?
006 210 057                 CMA             /RESTORE MEMORY
006 211 167                 LM A
006 212 302 247 006         JFZ INIU        /NOPE - THIS IS END OF RAM
006 215 043                 INX HL          /YUP - KEEP TRYING
006 216 303 204 006         JMP INIS
```

```
006 221 315 205 004 NUM,    CAL DEINT       /GET NUMERIC ARGUMENT
006 224 353                 DE<>HL          /REQUESTED ADDRESS -> HL
006 225 021 247 006         LXI DE MMEM     /MINIMUM POSSIBLE ADR -> DE
006 230 367                 CMPR            /REQUEST < MINIMUM ?
006 231 332 154 006         JTC INIT        /YUP - GIVE THE CHUMP ANOTHER CHANCE
006 234 053                 DCX HL          /FIRST LOC FOR STACK
006 235 176                 LA M            /GET CONTENTS
006 236 057                 CMA
006 237 167                 LM A            /WRITE IT BACK COMPLIMENTED
006 240 276                 CP M            /DID IT GO ?
006 241 057                 CMA
006 242 167                 LM A            /RESTORE CONTENTS
006 243 043                 INX HL
006 244 302 154 006         JFZ INIT        /NOPE - NO RAM WHERE HE SAYS
                        /
        247 006         MMEM=.              /LOWEST LOC FOR STACK RESET
                        /
006 247 042 377 004 INIU,   STHL SPRS       /YUP - MAKE IT THE STACK RESET ADR
006 252 041 154 006         LXI HL EOP      /BOTX IF HE SAYS "YES"
006 255 315 066 007         CAL WANT        /ASK "WANT SAVE / TAPE?"
006 260 123                 DSS "S,"A,"V,"E,"/,"T,"A,"P,"E,"?+200
006 261 101
006 262 126
006 263 105
006 264 057
006 265 124
006 266 101
006 267 120
006 270 105
006 271 277
006 272 315 054 007         CAL YSNO        /GET HIS ANSWER
006 275 041 137 000         LXI HL SNER     /HE SAID NO - POKE OUT TEST
006 300 042 175 005         STHL OSPK       /FOR SAVE / TAPE
006 303 041 002 006         LXI HL EOP3     /BOTX IF HE SAYS YES -> HL
006 306 315 066 007         CAL WANT        /ASK "WANT STRING I/O"
006 311 123                 DSS "S,"T,"R," ,"I,"/,"O,"?+200
006 312 124
006 313 122
006 314 040
006 315 111
006 316 057
006 317 117
006 320 277
006 321 315 054 007         CAL YSNO        /GET HIS ANSWER
006 324 041 156 001         LXI HL PEXP     /HE SAID NO, POKE OUT STRING PRINT
006 327 042 146 001         STHL PXPK
006 332 041 172 001         LXI HL NPRU     /POKE OUT STRING INPUT TEST
006 335 042 067 001         STHL SIPK
006 340 041 340 005         LXI HL EOP2     /BOTX IF HE SAYS YES -> HL
006 343 315 066 007         CAL WANT        /ASK "WANT ARRAYS? "
006 346 101                 DSS "A,"R,"R,"A,"Y,"S,"?+200
006 347 122
006 350 122
006 351 101
006 352 131
006 353 123
006 354 277
006 355 315 054 007         CAL YSNO        /GET HIS ANSWER
006 360 041 137 000         LXI HL SNER     /HE SAID NO, POKE OUT ARRAY LOOKUP
006 363 042 206 005         STHL LKP1+1     /MAKE IT A SYNTAX ERROR
006 366 042 213 005         STHL LKP2+1
006 371 303 003 007         JMP ICON        /CONTNUED AT ICON
                        /
        000 007             * PGE           /PUT IN JUMP TO BINL FOR BOOT
007 000 303 101 007         JMP BINL
                        /
007 003 041 341 001 ICON,   LXI HL LETS     /MAKE ARRAY ASSIGNMENT ILLEGAL
007 006 042 331 001         STHL DFPK
007 011 041 232 005         LXI HL EOP1     /THIS IS BOTX -> HL
007 014 257         INIV,   XR A            /DO A "NEW" COMMAND
007 015 167                 LM A            /BOTX WILL BE IN HL
007 016 043                 INX HL          /NOW
007 017 042 256 004         STHL BOTX       /SAVE IT
007 022 167                 LM A            /DO A NEW
007 023 043                 INX HL
007 024 167                 LM A
007 025 043                 INX HL
007 026 167                 LM A
007 027 042 260 004         STHL EOTX
007 032 041 204 000         LXI HL ENTR     /POKE OUT JMP TO INIT
007 035 042 001 000         STHL 1          /MAKE IT A JUMP TO ENTR
007 040 357                 MSGP            /PRINT SIGN ON MESSAGE
007 041 015                 DSS 15,12,"C,"A,"S,"U,"A,"L+200
007 042 012
007 043 103
007 044 101
007 045 123
007 046 125
007 047 101
007 050 314
007 051 303 302 007         JMP ICN2        /CONTINUED AT ICN2
                        /
                        /ROUTINE TO GET "Y" OR "N" ANSWER FROM TERMINAL.
                        /TZ MEANS "Y", FZ MEANS "N".
                        /
007 054 315 117 005 YSNO,   CAL TTYI        /GET HIS CHR
007 057 347                 DEVO            /ECHO IT
007 060 376 131             CPI "Y          /YES ?
007 062 312 014 007         JTZ INIV
007 065 311                 RET
007 066 357         WANT,   MSGP            /SR TO PRINT "WANT"
007 067 015                 DSS 15,12,"W,"A,"N,"T," +200
007 070 012
007 071 127
007 072 101
007 073 116
007 074 124
007 075 240
007 076 303 050 000         JMP MSG
                        /
        101 007             * PGE 101       /START ADR OF BINARY LOADER
                        /
                        /THIS SECTION POKES THE BINARY LOADER TO THE SAME I/O
                        /CONFIGURATION USED BY THE BOOTSTRAP LOADER AT ZERO.
                        /THIS IS EXECUTED ONLY ONCE, UPON ENTRY FROM THE
                        /BOOTSTRAP.  AFTER THE FIRST TIME EXECUTED, THE JUMP
                        /AT WORD 0 OF THE BINARY LOADER PAGE IS POKED TO JUMP
                        /AROUND THE I/O POKE.
                        /
007 101 072 007 000 BINL,   LDA 7           /INPUT STATUS PORT # -> A
007 104 062 271 007         STA RDIN+1      /POKE INPUT ROUTINE
007 107 052 011 000         LDHL 11         /STATUS MASK ->L, RFZ OR RTZ -> HL
007 112 174                 LA H            /CHANGE RTZ OR RFZ INTO JFZ OR JTZ
007 113 306 002             ADI 2
007 115 147                 LH A
```

```
007 116 042 273 007          STHL POK1+1     /POKE THE INPUT ROUTINE
007 121 072 014 000          LDA 14          /INPUT DATA PORT # -> A
007 124 062 300 007          STA POK2+1      /POKE THE INPUT ROUTINE
007 127 041 135 007          LXI HL REAC     /POKE OUT THE JUMP TO BINL
007 132 042 001 007          STHL PGE 1      /MAKE IT A JUMP TO READ-3
007 135 061 000 010 REAC,    LXI SP PGE+0256
007 140 016 000     READ,    LCI 0           /CLEAR CHECKSUM
007 142 315 270 007          CAL RDIN        /GET A CHR FROM TAPE
007 145 376 277              CPI 277         /IS IT AN EOT CHR ?
007 147 312 254 007          JTZ GOTO        /YUP - LOOK FOR START ADDRESS
007 152 376 377              CPI 377         /NOPE - IS IT A START OF BLOCK ?
007 154 302 140 007          JFZ READ        /NOPE - MUST BE LEADER, KEEP LOOKING
007 157 315 243 007          CAL ADIN        /GET THE LOAD ADDRESS -> HL
007 162 315 270 007          CAL RDIN        /BLOCK LENGTH -> A
007 165 267                  OR A            /BLOCK LENGTH = 0 ?
007 166 312 211 007          JTZ CKSM        /YUP - NO DATA, VERIFY CHECKSUM
007 171 137                  LE A            /MOVE BLOCK LENGTH -> E
007 172 315 270 007 DATA,    CAL RDIN        /GET A DATA BYTE FROM TAPE
007 175 167                  LM A            /PUT IT INTO MEMORY
007 176 276                  CP M            /DID IT WRITE PROPERLY ?
007 177 302 223 007          JFZ MERR        /NOPE - GIVE A CAN'T WRITE ERROR
007 202 201                  AD C            /UPDATE CHECKSUM -> A
007 203 117                  LC A            /UPDATED CHECKSUM -> C
007 204 043                  INX HL          /BUMP THE LOAD ADDRESS
007 205 035                  DCE             /DONE WITH THIS BLOCK YET ?
007 206 302 172 007          JFZ DATA        /NOPE - GET MORE DATA BYTES
007 211 315 270 007 CKSM,    CAL RDIN        /DONE WITH BLOCK, GET CHECKSUM -> A
007 214 271                  CP C            /DOES IT MATCH CALCULATED VALUE ?
007 215 312 140 007          JTZ READ        /YUP - LOOK FOR ANOTHER BLOCK
007 220 076 103              LAI "C          /NOPE - GIVE CHECKSUM ERROR
007 222 001                  DSS 1           /SETUP A BOGUS LXI BC INSTRUCTION
007 223 076 115     MERR,    LAI "M
007 225 323 001     ERR,     OUT 1
007 227 323 010              OUT 10
007 231 323 021              OUT 21
007 233 323 023              OUT 23
007 235 062 377 007          STA PGE 377
007 240 303 225 007          JMP ERR         /LOOP FOREVER
                    /
                    /THIS SUBROUTINE GETS TWO BYTES FROM TAPE INTO HL.
                    /
007 243 315 270 007 ADIN,    CAL RDIN        /GET FIRST BYTE
007 246 157                  LL A            /MOVE IT INTO -> L
007 247 315 270 007          CAL RDIN        /GET SECOND BYTE
007 252 147                  LH A            /MOVE IT INTO -> H
007 253 311                  RET
                    /
                    /COMES HERE WHEN EOT CHR IS FOUND.  IF A 100 BYTE FOLLOWS
                    /THE EOT, THE NEXT TWO BYTES ARE TAKEN TO BE A START ADDRESS
                    /CONTROL IS TRANSFERRED TO THIS ADDRESS.  IF NO 100 BYTE IS
                    /FOUND, WE ENTER AN INFINITE LOOP.
                    /
007 254 315 270 007 GOTO,    CAL RDIN        /GET A CHR FROM TAPE
007 257 376 100              CPI 100         /IS IT A 100 (OCTAL)
007 261 302 261 007 FORE,    JFZ FORE        /NOPE - JUMP HERE FOREVER
007 264 315 243 007          CAL ADIN        /START ADDRESS -> HL
007 267 351                  PC<HL           /INDIRECT JUMP TO START ADDRESS
                    /
                    /THIS SUBROUTINE FETCHES A CHR FROM THE INPUT DEVICE.
                    /THE CHR IS RETURNED IN THE A REG.  MUNCHES A & PSW.
                    /
007 270 333 005     RDIN,    INP 5           /INPUT READY STATUS -> A
007 272 346 001     POK1,    NDI 1           /MASK OFF UNNECESSARY BITS
007 274 312 270 007          JTZ RDIN        /JUMP IF NOT READY, KEEP TRYING
007 277 333 004     POK2,    INP 4           /IT'S READY - GET THE DATA -> A
007 301 311                  RET
        302 007     LLOC=.                   /SAVE ADDRESS OF LAST BYTE USED
                    /
007 302 357         ICN2,    MSGP            /CONTINUE SIGN ON MESSAGE
007 303 040                  DSS " ,"Y," ","."1,"6,15,212
007 304 126
007 305 040
007 306 056
007 307 061
007 310 066
007 311 015
007 312 212
007 313 052 377 004          LDHL SPRS
007 316 353                  DE<>HL          /LAST LOC -> DE
007 317 052 256 004          LDHL BOTX       /FIRST -> HL
007 322 173                  LA E            /DIFFERENCE -> HL
007 323 225                  SU L
007 324 157                  LL A
007 325 172                  LA D
007 326 234                  SB H
007 327 147                  LH A
007 330 315 066 004          CAL NOSP        /PRINT DIFFERENCE
007 333 357                  MSGP            /PRINT "BYTES FREE"
007 334 102                  DSS "B,"Y,"T,"E,"S," ,"F,"R,"E,"E+200
007 335 131
007 336 124
007 337 105
007 340 123
007 341 040
007 342 106
007 343 122
007 344 105
007 345 305
007 346 307                  RST             /RESET STACK AND ENTER
                    /
                    /THIS IS THE ROUTINE USED TO PUNCH MEMORY IN BOOTSTRAP FMT.
                    /
        000 011     *PGE+0512
011 000 061 000 012 MAKR,    LXI SP .+0256
011 003 006 377              LBI 377         /SEND 255 LEADER CHRS
011 005 076 302     MAKS,    LAI LLOC        /LEADER CHR -> A
011 007 315 126 006          CAL PNOU        /SEND A CHR OF LEADER
011 012 005                  DCB             /DONE WITH LEADER YET ?
011 013 302 005 011          JFZ MAKS        /NOPE - SEND SOME MORE
011 016 041 301 007          LXI HL LLOC-1   /HIGHEST ADR TO SENT -> HL
011 021 176          MAKT,   LA M            /GET A CHR TO PUNCH -> A
011 022 315 126 006          CAL PNOU        /PUNCH IT
011 025 055                  DCL             /PUNCHED IT ALL YET ?
011 026 302 021 011          JFZ MAKT        /NOPE - KEEP SENDING
011 031 176                  LA M            /SEND LAST CHR
011 032 315 126 006          CAL PNOU
011 035 303 000 374          JMP '374        /ALL DONE , BACK TO MONITOR
                    $

00ERRORS
                    /
                    /THIS SECTION OF CODE IS THE I/O POKE SECTION.
                    /IT IS READ IN BY THE BINARY LOADER AFTER THE I/O
                    /SECTION OF CASUAL.  THIS ROUTINE READS THE FRONT PANEL
                    /SENSE SWITHCES AND POKES THE I/O FOR SOME MITS I/O
                    /BOARDS.  THE SWITCHES MEAN THIS WHEN THEY ARE UP.
                    /
                    /A15    MITS ACR FOR SAVE/TAPE COMMANDS
                    /A14    MITS SIOA, B, C  REV 0 FOR TERMINAL
                    /A13    MITS 88-PIO FOR TERMINAL
                    /A12    RESERVED FOR MITS 4PIO (NOT FUNCTIONAL NOW)
                    /A11    RESERVED FOR MITS 2SIO (NOT FUNCTIONAL NOW)
                    /A10    RESERVED FOR MITS 2SIO (NOT FUNCTIONAL NOW)
                    /A9     CONTROL LOGIC STANDARD I/O
                    /A8     USE CUSTOM I/O FROM LOCATION 50
                    /NONE   MITS SIOA, B, C  NOT REV 0 FOR TERMINAL
                    /
                    /THIS LOADS AT LOCATION 100,  AND WHEN IT IS DONE POKING
                    /IT GOES BACK TO THE BINARY LOADER WHICH WILL THEN READ
                    /CASUAL IN ON TOP OF THIS.
                    /
        100 000             *100
000 100 041 000 007 IOPA,    LXI HL PGE      /PUSH ADR OF BINARY LOADER
000 103 345                  PSHX HL
000 104 333 377              INP 377
000 106 346 002              NDI 2           /TEST A9
000 110 300                  RFZ             /HE WANTS CONTROL LOGIC - ALL DONE
000 111 333 377              INP 377
000 113 267                  OR A            /TEST A15
000 114 362 160 000          JFS IOPB        /DOESN'T HAVE AN ACR
000 117 076 006              LAI 6
000 121 062 143 006          STA CHIN+1
000 124 062 130 006          STA PNOV+1
000 127 074                  INA
000 130 062 152 006          STA CIDP+1
000 133 062 140 006          STA CODP+1
000 136 076 302              LAI 302
000 140 062 146 006          STA CIA
000 143 062 133 006          STA COA
000 146 076 001              LAI 1
000 150 062 145 006          STA CIRM+1
000 153 076 200              LAI 200
000 155 062 132 006          STA CORM+1
000 160 333 377     IOPB,    INP 377
000 162 346 100              NDI 100         /TEST A14
000 164 312 235 000          JTZ IOPC        /DOESN'T HAVE SIOA, B, OR C
000 167 257                  XR A
000 170 062 135 005          STA TRDY+1
000 173 062 042 000          STA DEVQ+1
000 176 074                  INA
000 177 062 126 005          STA TIDP+1
000 202 076 040              LAI 40
000 204 062 137 005          STA TIRM+1
000 207 062 113 005          STA TODP+1
000 212 076 002              LAI 2
000 214 062 044 000          STA TORM+1
000 217 076 312              LAI 312
000 221 062 122 005          STA TIA
000 224 062 106 005          STA DEVP
000 227 076 310              LAI 310
000 231 062 163 005          STA TIB
000 234 311                  RET
000 235 333 377     IOPC,    INP 377
000 237 346 040              NDI 40          /TEST A13
000 241 312 307 000          JTZ IOPD        /DOESN'T HAVE AN 88-PIO
000 244 257                  XR A
000 245 062 135 005          STA TRDY+1
000 250 062 042 000          STA DEVQ+1
000 253 074                  INA
000 254 062 126 005          STA TIDP+1
000 257 062 044 000          STA TORM+1
000 262 062 113 005          STA TODP+1
000 265 074                  INA
000 266 062 137 005          STA TIRM+1
000 271 076 312              LAI 312
000 273 062 122 005          STA TIA
000 276 062 106 005          STA DEVP
000 301 076 310              LAI 310
000 303 062 163 005          STA TIB
000 306 311                  RET
000 307 333 377     IOPD,    INP 377
000 311 346 177              NDI 177         /ALL LOW ? (A15 DON'T CARE)
000 313 302 362 000          JFZ IOPE        /DOESN'T HAVE SIOA, B, OR C NOT REV0
000 316 257                  XR A
000 317 062 135 005          STA TRDY+1
000 322 062 042 000          STA DEVQ+1
000 325 074                  INA
000 326 062 137 005          STA TIRM+1
000 331 062 126 005          STA TIDP+1
000 334 062 113 005          STA TODP+1
000 337 076 200              LAI 200
000 341 062 044 000          STA TORM+1
000 344 076 300              LAI 300
000 346 062 163 005          STA TIB
000 351 076 302              LAI 302
000 353 062 122 005          STA TIA
000 356 062 106 005          STA DEVP
000 361 311                  RET
000 362 333 377     IOPE,    INP 377
000 364 346 001              NDI 1           /TEST A8
000 366 310                  RTZ             /DOESN'T WANT CUSTOM
000 367 041 050 000          LXI HL CUST     /START OF CUSTOM SAVE AREA
000 372 176                  LA M
000 373 062 135 005          STA TRDY+1
000 376 043                  INX HL
000 377 176                  LA M
001 000 062 137 005          STA TIRM+1
001 003 043                  INX HL
001 004 176                  LA M
001 005 062 122 005          STA TIA
001 010 075                  DCA
001 011 075                  DCA
001 012 062 163 005          STA TIB
001 015 043                  INX HL
001 016 176                  LA M
001 017 062 126 005          STA TIDP+1
001 022 043                  INX HL
001 023 176                  LA M
001 024 062 042 000          STA DEVQ+1
001 027 043                  INX HL
001 030 176                  LA M
001 031 062 044 000          STA TORM+1
001 034 043                  INX HL
001 035 176                  LA M
001 036 062 106 005          STA DEVP
001 041 043                  INX HL
001 042 176                  LA M
001 043 062 113 005          STA TODP+1
001 046 043                  INX HL
001 047 176                  LA M
001 050 062 143 006          STA CHIN+1
001 053 043                  INX HL
```

```
001 054 176                 LA M
001 055 062 145 006         STA CIRM+1
001 060 043                 INX HL
001 061 176                 LA M
001 062 062 146 006         STA CIA
001 065 043                 INX HL
001 066 176                 LA M
001 067 062 152 006         STA CIDP+1
001 072 043                 INX HL
001 073 176                 LA M
001 074 062 130 006         STA PNOV+1
001 077 043                 INX HL
001 100 176                 LA M
001 101 062 132 006         STA CORM+1
001 104 043                 INX HL
001 105 176                 LA M
001 106 062 133 006         STA COA
001 111 043                 INX HL
001 112 176                 LA M
001 113 062 140 006         STA CODP+1
001 116 052 070 000         LDHL OSLO
001 121 042 200 005         STHL OSPK+3
001 124 311                 RET
```

```
                /FUN-   CONTROL SIOA    SIOA    88-PIO  4PIO    2SIO    MNEU-
                /CTION  LOGIC           REV0                            MONIC
                /-------------------------------------------------------------
                /TIRP   1       0       0       0       20      20      TRDY+1
                /TIRM   1       1       40      2       100     1       TIRM+1
                /TIA    312     302     312     312     312     312     TIA
                /                                                       -2 TIB
                /TIDP   0       1       1       1       21      21      TIDP+1
                /TORP   1       0       0       0       20      20      DEVQ+1
                /TOA    312     302     312     312     312     312     DEVP
                /TODP   10      1       1       1       23      21      TODP+1
                /CIRP   5       6                                       CHIN+1
                /CIRM   1       1                                       CIRM+1
                /CIA    312     302                                     CIA
                /CIDP   4       7                                       CIDP+1
                /CORP   5       6                                       PNOV+1
                /CORM   2       200                                     CORM+1
                /COA    312     302                                     COA
                /CODP   16      7                                       CODP+1
                /
                /FUNCTION KEY:
                /
                /FIRST LETTER:
                /T=     TERMINAL
                /C=     CASSETTE
                /
                /SECOND LETTER:
                /I=     INPUT
                /O=     OUTPUT
                /
                /LAST TWO LETTERS:
                /RP=    READY PORT
                /RM=    READY MASK
                /A=     ACITVE, HIGH OR LOW
                /DP=    DATA PORT
                /
                /
                /IF YOU ARE USING AN I/O BOARD NOT SUPPORTED BY CASUAL,
                /IT IS BEST TO USE THE CUSTOM I/O PROCEDURE.  AFTER LOADING
                /THE BOOTSTRAP, LOAD THE CUSTOM I/O TABLE (SEE BELOW).
                /SET SWITCH A8 UP, THE REST DOWN, AND EXECUTE THE BOOT
                /AS USUAL.
                /
                /LOCATION       CONTENTS
                /050            TIRP
                /051            TIRM
                /052            TIA
                /053            TIDP
                /054            TORP
                /055            TORM
                /056            TOA
                /057            TODP
                /060            CIRP
                /061            CIRM
                /062            CIA
                /063            CIDP
                /064            CORP
                /065            CORM
                /066            COA
                /067            CODP
                /070            OS ADDRESS LOW
                /071            OS ADDRESS HIGH
                /
        000 007 PGE='7
        050 000 CUST=50
        175 005 OSPK='5 175
        070 000 OSLO=70
        134 005 TRDY='5 134
        136 005 TIRM='5 136
        122 005 TIA='5 122
        163 005 TIB='5 163
        125 005 TIDP='5 125
        041 000 DEVQ='0 41
        043 000 TORM='0 43
        106 005 DEVP='5 106
        112 005 TODP='5 112
        142 006 CHIN='6 142
        144 006 CIRM='6 144
        146 006 CIA='6 146
        151 006 CIDP='6 151
        127 006 PNOV='6 127
        131 006 CORM='6 131
        133 006 COA='6 133
        137 006 CODP='6 137
                /
                $

00ERRORS
```

A thought, compliments of David G.: If we put an automatic disc ejector on a disc drive, then perhaps we could put those old Wurlitzer boxes to some interesting use.

ACTIVE LOW by Mark Space

Some signals on a microcomputer's bus are normally low and go high when they are meaningful. These are called "active high". Some lines are just the opposite and they are called "active low".

The commonly accepted notation for an active low line is to put a bar "‾‾‾" over the line's name, for example: $\overline{STB}$. Now, when this is read you say "not strobe" or sometimes "bar strobe."

With this in mind then we thought that the following active low signals might be useful additions to a microcomputer's bus structure.

$\overline{AWK}$ — This signal is present during the hours of 2:00 and 9:00 a.m.

$\overline{HOM}$ — This line goes low when you leave the house—useful in intelligent security systems.

$\overline{NOW}$ — Present when interrupts are disabled.

$\overline{IN}$ — This line goes low when the system is out for lunch.

$\overline{NOT}$ — This form of interrupt is acknowledged by the "Who's There?" line.

$\overline{HOL}$ — This signal is present when a ram board goes out in the middle of your memory map.

$\overline{RT}$ — Signifies that your jump went the wrong way.

$\overline{MINE}$ — In a polled interrupt system all devices not requesting service must pull this line low.

$\overline{T}$ — This line goes low when certain combinations of ASCII data appear, usually in groups of four. Also used in some systems in response to certain types of graphic images.

$\overline{NCE}$ — Interchanged in some systems with $\overline{T}$.

$\overline{KDNG}$ — Data is valid now.

$\overline{TNITE}$ — This signal is issued when the system has a headache.

$\overline{ENF}$ — This line goes low if the second byte of the op code is missing or in some systems this may signify that all available memory has been used.

$\overline{BZI}$ — Issued to DMA devices to let them know they can have the bus.

$\overline{TRYN}$ — This signal is only present during NOP's.

2B or $\overline{2B}$ — There is much debate over whether or not this line should be 2B or $\overline{2B}$, but that is the question.

$\overline{ALWD}$ — Issued in response to an illegal op code or in some systems it is $\overline{ALOUD}$ meaning "turn the TTY off; video display only".

$\overline{NTRSTD}$ — Issued by Selectrics usually in conjunction with the $\overline{MYTYPE}$ signal.

$\overline{ON}$ — The switch is off.

$\overline{HEAD}$ — Processor's response to the programmer who jumps to the second byte of the op code.

$\overline{LIT}$ — Signal goes low when all the LED's on the front pan panel are out.

$\overline{ME}$ — In a multi-processing environment the control processor polls the slave processors asking who put that data into the common memory and all processors that didn't will issue this signal.

$\overline{UP}$ — This line goes low when the system is crashed.

$\overline{HERE}$ — In Z-80 systems this line goes low during each cycle of a block search when the data is not found. (In some systems this is the $\overline{FND}$ line).

$\overline{BYNIT}$ — Invalid or spurious data.

$\overline{BAD}$ — During each cycle of a ram test this line will normally be low unless a location is faulty.

$\overline{GLTY}$ — Response to the "How do you plead" line. Also low on computers owned by hobbyists who disagree with Bill Gates.

$\overline{THERE}$ — Signifies that the TTY connector has come undone.

$\overline{BIT}$ — This signal is present when a hobbyist doesn't have his own computer yet.

$\overline{FNI}$ — Issued in response to most of the foregoing lines.

These are but a few of the lines that you might find useful in your own computer system and we're sure you can imagine a lot more that we didn't cover here.

Oh, yes. One more. There always are empty lines on everyone's bus, and these of course are then designated: $\overline{USED}$.

A REPLY: STRUCTURED PROGRAMMING

Dear Sir, Oct. 23, 1976

Several months ago, I wrote a letter to *DDJ*, parts of which appeared in Vol. 1, No. 6, page 40. In Vol. 1, No. 9, page 37, Tim Bonham takes issue with several of the comments in my letter. I wish to have the opportunity to in turn take issue with several of Mr. Bonham's comments.

First, Mr. Bonham states that I seem to equate structured programming with lots of control structures. As a matter of fact, I don't make such as equation since I am aware that SP (structured programming) involves concepts other than control structures such as top down programming and so forth. However, SP in my mind does involve lots of control structures, and I will explain why later. For the purposes of this letter, I will assume that SP is concerned only with controlled structures.

Second, Mr. Bonham states that one of the important events in the history of structured programming was the publication of a proof that all programs could be written using only three control structures, namely, sequence, if-then-else, and loops. I assume that Mr. Bonham is referring to a proof which appeared in a paper written by Boehm and Jacopini [1]. Knuth [2] says the following about this result:

> **Recent interest in structured programming has caused many authors to cite Jacopini's result as a significant breakthrough and as a cornerstone of modern programming technique. Unfortunately, these authors are unaware of comments made by Cooper . . . and later by Bruno and Steiglitz . . ., namely, that from a practical standpoint the theorem is meaningless.**

Knuth goes on to show how Jacopini's result may be used to put any program into a virtually structureless form.

Third, Mr. Bonham states that one of the basics of SP is the use of only a very few control structures. If this is indeed the case, then we can do much better than SIL (sequencing, if-then-else, and looping), because, as Presser [3] has shown, if-then-else is superfluous. However, not even Presser advocates the complete elimination of if-then-else. Thus, I believe that it can be said that the minimum feasible set of control structures is not the same as the minimum practical set of control structures; although, there does seem to be a general consensus that a minimum practical set must include SIL. Where one draws the line beyond SIL seems to be strictly a case of *chacun a son gout.* Zahn [4] for example, seems to feel that even adding the FOR statement and recursive subprograms is not enough. Vaughan [5] argues for including both the labelled and indexed CASE statement. A casual examination of the literature will reveal various proposals for control structures to supplement SIL. In the absence of any precise, generally accepted definition of SP, I am inclined to believe that a minimum practical set of control structures will include substantially more than SIL.

Fourth, Mr. Bonham states that I seem to consider PL/I to be a simple SP language, and that he does not consider PL/I to be an SP language. I do not wish to refute Mr. Bonham's claim that PL/I is not an SP language, since I am in sympathy with his view on this issue. However, there are any number of textbooks with titles like "Structured Programming in PL/I" which suggests that PL/I is being treated as an SP language whether in fact it is or not. This together with the fact tnat PL/I is a commonly used complex language is the reason why I cited it in my letter.

Yours,

Fred J. Dickey
3420 Granville Rd
Westerville OH 43081

[1] Boehm, C. and Jacopini, G. "Flow-diagrams, Turing Machines, and Languages with only two formation rules," *CACM,* Vol. 9, No. 5, 1966, pp. 366-71.

[2] Knuth, D. "Structured Programming with GOTO Statements," *Computing Surveys,* Vol. 6, No. 4, 1974, pp. 261-301.

[3] Presser, L. "Structured Languages," *Sigplan Notices,* Vol. 10, No. 7, 1975, pp. 22-24.

[4] Zahn, C., Jr. "Structured Control in the Programming Languages," *Sigplan Notices,* Vol. 10, No. 7, 1975, pp. 13-15.

[5] Vaughan,W. C..M.. "Another Look at the CASE Statement," *Sigplan Notices,* Vol. 9, No. 11, 1974, pp. 32-36.

COMPUTER-BASED INSTRUCTIONAL SYSTEMS MEETING

The 1977 Winter Meeting of the Association for the Development of Computer-Based Instructional Systems (ADCIS) will be held in Newark, DE, February 22-24, 1977. For further information, contact the conference host, Fred Hofstetter, Dept. of Music, Univ. of Delaware, Newark, DE 19711, (302) 738-2497.

SOME SOFTWARE NEEDS & HOW TO FILL 'EM

1) It seems to me that what the microcomputer market needs most right now are good software development tools. High priority includes:
   a) a good monitor/operating system: one such as described in *DDJ* (April 1976) is on the right track. Sphere's new DOS also seems very promising. DOS not only allows user-cataloged 32-character file names, but also has a number of very useful monitor requests built in, which take care of all the I/O interfaces for the user.
   b) a macro-assembler that can run resident on a microsystem. The need for this should be self-explanatory.
   c) a *simple* procedure-oriented language that can be compiled by a resident compiler & can interface with assembly language subroutines. TINY HI looks good in this respect. The idea here is that system development is just too slow if done in assembly language. Furthermore, assembler listings are very difficult to decipher as to control structures and such. A good procedure-oriented language will provide the 3 or 4 basic control structures plus *as little else as can be gotten by with.* "Efficiency" can well be sacrificed for gains in readability, understandability and maintainability.
2) I would like to encourage *you* to encourage manufacturers to seek out a few top-notch software types and turn them loose for a few weeks. Promise them a bonus for early completion and institute a penalty clause for late delivery – but get a high-level language compiler out and get it out *fast.* Once a tool like TINY HI is in the hands of a large number of people, then you'll see some progress. Assembly language is indispensable *for certain tasks,* but for the bulk of application-programming, it continues to be a millstone around our necks!

Larry E. Walker

CAL INTERPRETER PROPOSED

Dear Jim,

I just got Niklaus Wirth's book *Systematic Programming: An Introduction* (Prentice-Hall, 1973), which is about PASCAL (but not mentioned in the title). I sure had a time finding anything on PASCAL, even in N.Y.C.

I notice that it is ALGOL like in many ways, but it does pick up some of the JOSS and CAL flavor.

This leads me to wonder why *Dr. Dobb's* is not looking into trying to get CAL interpreters or compilers started.

Joseph F. Gaffney
321 Lyndhurst Ave.
Lyndhurst, NJ 07071

[Great idea! How 'bout sending us a CAL interpreter for some micro, in the next month or two?–Editor]

HORRORS!–WE LEFT SOME ADDRESSES OUT OF ARTICLES IN PREVIOUS ISSUES

Here they are:

Itty Bitty Computers & Tom Pittman
Box 23189
San Jose, CA 95153
(408) 578-4944
[October issue. 6800 and 6502 Tiny BASIC for $5]

Per Sci
4087 Glenoe Ave.
Marina del Rey, CA 90291
(213) 822-7545
[August issue. Dual-drive floppy disc drive for $1K with *fast* voice-coil head positioner]

NIBL -- Tiny Basic for National's SC/MP Kit

complete documentation & annotated source code

by Mark Alexander, National Semiconductor Corp.
Nov. 29, 1976

Introduction

NIBL (National Industrial Basic Language) is a machine-oriented programming language for the SC/MP. It is a language similar to Tiny BASIC, but it also has some unique features. Many of these features, such as a genuinely useful control structure (the PASCAL-influenced DO/UNTIL) and the indirect operator ("@") have been added to the language to allow NIBL to be nearly as flexible as machine language in such applications as medium-speed process control.

By using NIBL, one trades the high execution speed and low memory consumption of machine language for some very tangible advantages: Program readability, modifiability, and reliability, which are truly difficult to achieve in machine language programs.

NIBL programs are interpreted by a large (4K byte) SC/MP program that resides in ROM. The interpreter is broken into two blocks: a program written in an Intermediate (or Interpretive) Language – I. L. for short – which does the actual interpretation; and a collection of SC/MP machine language sub-routines invoked by the I. L. program. The I.L. approach is well-documented in Vol. 1, No. 1 of *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia,* and readers should refer to that issue for a more detailed description of the interpretation process.

In Table 1, the formal grammar for NIBL is given. This is the ultimate authority (other than the interpreter itself) on how legal NIBL statements are formed. The following descriptions of the NIBL statements will refer to portions of the grammar. Table 2 contains a list of the error message produced by the NIBL system. Finally, a listing of the interpreter is given in the Appendix.

History of NIBL

NIBL came into this world as an interpreter for Tiny BASIC, as originally described in the first issue of *Dr. Dobb's Journal.* That program was written by Steve Leininger, who subsequently left before the program was ever assembled or executed. The current version of NIBL is an almost complete re-write of the original interpreter, with changes and additions being made to improve the modularity of the program, to greatly increase execution speed, and to extend the capabilities of the language itself.

The program was developed on the PACE Disk Operating System, and was assembled by a PACE-resident cross-assembler for the SC/MP.

System Requirements

The NIBL interpreter is intended to be a ROM-resident program in the first 4K of the SC/MP address space (although it will run just as well in RAM). The interpreter requires at least 2K bytes of RAM starting at address 1000 (base 16), of which the interpreter uses nearly 300 bytes for stacks, variables, etc., leaving the rest for the user's program. Another 2K bytes of memory may be added to fill up this 4K page, forming what is hereafter referred to as "Page 1".

The SC/MP architecture forces memory to be split into pages of 4K bytes each; therefore, NIBL allows seven such pages to be used for storing programs. NIBL programs in the seven pages are edited separately, but may be linked together during program execution by special NIBL statements described below. The first page, mentioned above, must be RAM since the interpreter uses part of it as temporary storage; the part used to store programs starts at location 111E (base 16).

The other six pages, each of which starts at location n000 (base 16), where n is the page number, may be either RAM or ROM. Page 2 is a special page: it can contain a NIBL program to be executed immediately upon powering up the NIBL system.

The memory organization of NIBL is shown in Figure 1.

Throughout this article, the assumption is made that the user has a teletype with paper tape reader and punch, as with the SC/MP Low Cost Development System. In fact, NIBL was designed to use the LCDS teletype interface, but to be completely independent of the LCDS LCDS firmware. If NIBL is to be run on its own, the system should have the same configuration for the teletype, with the reader relay being operated directly by the SC/MP. At present, paper tape is the only medium for saving NIBL programs, but as soon as the hardware and software for a SC/MP cassette interface become availalbe, NIBL will be able to link to routines for saving and loading programs with ease.

Since the teletype interface is not based on a UART, the terminal baud rate can only be changed by modifying the timed delays in NIBL's I/O routines. NIBL has been run successfully at 1200 baud with a CRT terminal; the listing of the program in the Appendix is for a 110 baud system.

Communicating with NIBL

When the NIBL system is ready to accept input, it prompts at the teletype with a ">" sign. (NIBL is now in "edit mode".) The user then enters a line terminated by a carriage return. There are several special characters that are used to edit lines as they are typed:

Shift/0 (back arrow) cuases the last character typed to be deleted.

Control/U (echoes as " U") causes the entire line to be deleted; NIBL reprompts for a new line.

Entering a line to NIBL without a leading line number causes the line to be executed directly by NIBL. Most NIBL statements, as well as the four program control commands, may be executed in this manner.

A line with a leading number (in the range 0 through 32767) is entered into the NIBL program in the current page. (Make sure that the value of the pseudo-variable PAGE is valid, so that the line isn't lost into non-existent memory.) The NIBL editor sorts the program lines as they are entered into ascending order by line number.

Typing a line number followed by a carriage return deletes that line from the program. Typing a line with the same number as an existing line's causes the new line to replace the old one in the program.

Each of the seven memory pages may contain a different program, separate from the rest. Editing the program in one page will not affect the other pages. To switch editing from one page to another, simply type PAGE = n, where n is the number of the new page.

Variables

There are twenty-six variable names in NIBL: the letters A through Z. They are all 16-bit binary variables, so they can be used to hold addresses as well as signed numeric data. The variables are already pre-declared for the user, and space is allocated for them in RAM when NIBL powers up.

Constants

NIBL allows either decimal or hexadecimal (base 16) constants to appear in expressions. Decimal constants must lie in the range 0

through 32767; the unary minus ("–") is used to obtain negative values. The value –32768 is a valid NIBL integer, but it is not legal as it stands. To represent it, use –32767–1 or #8000 instead.

Hexadecimal constants are denoted by a pound sign ("#") followed by a string of hexadecimal digits (0-9, A-F). NIBL does not check for overrun in hex constants; consequently, only the 4 least significant digits of the nex digit string are kept.

Functions

NIBL provides three built-in functions that may appear in any expression. These are described as follows:

RND (X, Y) returns a pseudo-random integer in the range X through T, inclusive, where X and Y are arbitrary expressions.
T, inclusive, where X and Y are arbitrary expressions. In order for the function to work properly, the value of Y – X should be positive and no greater than 32767.

MOD (X, Y) returns the absolute value of the remainder from X divided by Y (where X and Y are expressions).

TOP (with no arguments) returns the address of the first free byte in the memory page currently being edited or executed. In other words, it is the address of the top of the NIBL program in the current page, plus one.

Pseudo-variables

NIBL has two pseudo-variables in addition to the standard variables. These are STAT and PAGE. Both of these varialbes may appear on either side of an assignment statement.

STAT represents the SC/MP status register. The current value of the status register can be referred to by using STAT in an expression; or an assignment may be made to the status register by executing a statement such as STAT = 4 or STAT = STAT OR #20. When NIBL makes an assignment to the status register in this manner, it clears the interrupt-enable bit of the value before it is actually assigned. Note also that only the lower byte of the value is assigned; the high byte is ignored.

The carry and overflow bits in STAT are meaningless since the NIBL system is continually modifying them. The utility of STAT lies in the fact that 5 of its bits are connected to I/O sense lines on the SC/MP chip.

The pseudo-variable PAGE contains the number of the memory page currently being executed or edited. As indicated in Figure 1, there are seven pages in which NIBL programs may be stored; therefore, PAGE may lie only in the range 1 through 7. If an assignment of a value outside this range is made to PAGE, only the 3 least significant bits of the value are used – and zero is automatically changed to one.

If PAGE is modified while NIBL is in edit mode, all subsequent editing will take place in the new page.

If PAGE is modified by a NIBL program during execution, control will be passed to the first line of the NIBL program in the new page. This transfer would be effected by a statement such as PAGE = 6 or PAGE = PAGE + 1. Thus, several NIBL programs residing in different 4K pages may be linked together as one large program, if need be. This would allow one to write a 28K STAR TREK program in NIBL, a Herculean and indeed foolish task.

Control may also be transferred from one page to another by three other statements: RETURN, NEXT, and UNTIL. Thus, the first part of a subroutine or loop may be in one page, and the second part may be in another (with control being transferred between the two parts by an assignment to PAGE). In these three special cases, NIBL automatically updates the value of PAGE as the statements are executed.

Relational Operators

NIBL provides the standard BASIC relational operators, for comparing the values of integer expressions. The operators are as follows:

=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	not equal to
<	less than
>	greater than

All of these operators produce 1 as a result if the relation is true, and 0 if the relation is false. Note that the relational operators may appear anywhere that an expression is called for in the NIBL grammar, not only in IF statements.

Arithmetic Operators

NIBL provides the four standard arithmetic functions: addition (+), subtraction or unary minus (–), multiplication (*), and division (/). Since only integers are allowed in NIBL, all quotients are truncated (the MOD function can be used to obtain remainders from division). Any overflow or underflow (other than division by zero) is ignored by NIBL; the reasoning behind this is that it may often be necessary to treat NIBL expressions as unsigned values, such as when performing calculations using memory addresses as the operands. Thus the value of 32767 + 1 is –32768 (or in hexadecimal, #7FFF + 1 = #8000, which makes more sense).

Logical Operators

In NIBL, there are three logical operations that may be performed on values: AND, OR, and NOT. The first two are binary operators, and the latter is unary. All three perform bitwise logical operations on 16-bit arguments, producing 16-bit results. AND, OR, and NOT are sufficient to simulate any other logical operation, through various combinations of the operators.

The Indirect Operator

The indirect operator "@" realizes the functions of PEEK and POKE operations in other BASICs, but with somewhat more elegance. The "@" sign followed by an address (whcih can be a constant, variable, or expression in parentheses) denotes the contents of that address in memory. Thus, if memory location 245 (decimal) contains 60, the statement X = @245 would result in the value 60 being assigned to X. The indirect operator may also appear on the left side of an assignment statement. For example, @X=@(Y+10) would result in the memory location pointed to by X being assigned the value of the memory location pointed to by the value Y+10.

Despite this, it is still safest to use plenty of parentheses in expressions to make the intent clear.

Use of the indirect operator is not limited to reading from or writing to memory: it also provides a simple way to communicate with peripheral devices that are interfaced to the SC/MP through memory addresses. Note that the "@" operator can only access memory one byte at a time, and that when an assignment is made to a memory location, only the low order byte of the value is moved to the location; the high order byte is ignored.

The indirect operator can also be used to simulate arrays in NIBL. For example, if we wish to define an M x N matrix of one-byte positive integers, we can access the (I,J)th element of the matrix (assuming that (0,0) is a legal element in the matrix) with the expression @(A+I*N+J). An assignment could be made to that same element by placing the expression on the left side of an assignment statement.

Expressions

Expressions in NIBL are made up of the components described above: variables, constants, function references, pseudo-variables, and operators binding them all together. NIBL expressions are all 16-bit integers. Evaluation of expressions takes place left-to-right, and the order in which operations take place is determined by operator precedence and the presence of parentheses. The order of evaluation can be deduced from the grammar in Table 1; here is a table of operator precedence:

Lowest precedence (applied last): <, >, <=, >=, =, <>
+, –, OR
*, /, AND
Highest precedence (applied first): @, NOT

Program Control Commands

LIST causes the entire program in the current page to be listed. Listing can be halted by hitting any key on the teletype: the BREAK key works best.

LIST <number > causes listing to begin at the given line number (or the nearest one greater than the number), rather than at the first line.

LISTing a program is the method used to save it on paper tape. To accomplish this, type LIST with the punch off, then turn on the punch and hit carriage return. After the program is dumped, type a Shift/0 with teletype on LOCAL so that the last character (a " >") will be deleted when the tape is entered to NIBL at a later time. NIBL will accept a tape made in this fashion at any time during edit mode. The tape reader is enabled at all times by NIBL, and it does not distinguish between the reader and the keyboard when accepting input. Superfluous line-feed and null characters on the tape are echoed but ignored.

RUN causes three actions: first, all variables are zeroed; secondly, all stakes (the FOR, DO, and GOSUB stacks) are cleared; and finally the program in the current page is executed, starting with the first line in sequence.

RUN is not the only way to start program execution: GOTO and GOSUB can slo be used to jump into a program from edit mode. For example, if there is a subroutine at line 1000 that is being tested, typing GOSUB 1000 will cause that routine to be executed, with NIBL returning to edit mode upon encourntering a RETURN statement. When GOTO and GOSUB are used to run a program, the variables and stacks are not cleared.

Hitting any key while a program is being run will cause NIBL to break execution, printing a message and the line number where the break was detected. The BREAK key on the teletype works best for this.

CLEAR causes all variables to be zeroed and the three stacks mentioned above to be cleared. This latter feature of the CLEAR command

is quite useful after a stack nesting error has occurred (for example, if GOSUBS are nested more than eight levels deep).

NEW clears the programs in Page 1, and changes the value of PAGE to 1. This is the form of the command most likely to be used by NIBL novices who do not wish to be confused by the page selection features of NIBL. NEW should be the first thing one types in to NIBL when first powering up.

NEW <number> sets the value of PAGE to the <number>, and clears the program in that page.

Assignment Statements

Already, two different types of assignment statements have been mentioned: assignments to the pseudo-variables STAT and PAGE, and assignments to memory locations with the indirect operator. Another form of the assignment statement is the conventional assignment to a variable (A – Z), e.g. A=A + ! or A = 32 <((4 * I). There are also statements which look like string assignments, but there are not standard BASIC, and are described later in the section on string handling. The word "LET" is optional in front of any assignment statement (leaving it out increases execution speed, unlike most Tiny BASIC systems).

If/then Statement

The IF statements allows conditional execution of one or more statements (as many as can fit on one line). The syntax for the IF statement is:

'IF' Rel-exp 'THEN'? Statement

which indicates that the word THEN is optional, and that any statement (including another IF statement) may follow the conditional expression. If the IF condition is true (i.e. is non-zero), the statement following it (and any others on the line) will be executed; otherwise, control immediately transfers to the next program line. The condition does not need to contain relational operators: a statement such as IF MOD (A,5) THEN.... is perfectly legal. In this example, the statement following the THEN would be executed if A were not divisible by 5.

GOTO, GOSUB, AND RETURN STATEMENTS

The syntax for the GOTO statement is 'GOTO' followed by an expression. The effect of the GOTO statement is to transfer control to the line whose number is indicated by the expression. An error occurs if the specified line does not exist in the current page. Unlike standard BASICs, any arbitrary expression can be used to specify the line number, as well as the usual decimal constant. This allows computed branches to be performed with the same effect as the ON . . . GOTO statement in standard BASIC.

The GOSUB statement is identical to the GOTO statement in form. It too causes a branch to a new line, but it also saves the address of the following statement on a stack. When a RETURN statement is executed, the saved address is popped from the stack, and control returns to that point in the program. Since an actual address, not a line number, is saved on the GOSUB stack, GOSUB statements may appear anywhere on a multiple-statement line.

GOSUBs may be nested up to eight levels deep; an error will occur if an attempt is made to exceed this limit. The error condition does not destroy the previous contents of the stack, so a RETURN statement can be executed (even in edit mode) without an error occurring. However, any modification of the NIBL program will clear the GOSUB stack, so that a subsequent RETURN without a GOSUB will cause an error.

DO AND UNTIL STATEMENTS

The DO and UNTIL statements are useful in writing program loops efficiently, without using misleading GOTO statements. Enclosing a group of zero or more statements between a DO statement and an UNTIL <condition> statement (where <condition> is an arbitrary expression) will cause the statement group to be repeated one or more times until the <condition> becomes true (i.e., non-zero). As an example of the use of the DO and UNTIL statements, we present a program that prints the prime numbers:

```
 10 PRINT 1: PRINT 2
 20 I=3
 30 DO
 40       J=I/2: N=2
 50       DO
 60          N=N+2
 70       UNTIL (MOD(I,N=0) OR (N>J)
 80       IF N>J PRINT I
 90       I=I+2
100 UNTIL 0
```

DO loops may be nested up to eight levels deep, and NIBL acts in the same manner if an overflow occurs as it does with a GOSUB overflow. NIBL also reports an error if an UNTIL statements occurs without a previous DO. A single DO loop may have more than one UNTIL statement as a terminator. For example, if one wished to exit abnormally out of a DO loop and transfer to some appropriate line, it could be done in the following manner:

UNTIL 1: GOTO X

where X is the line number.

Neither the DO nor the UNTIL statement may be executed in edit mode.

FOR AND NEXT STATEMENTS

The NIBL FOR statement is virtually identical to that in standard BASICs; consequently, it is not explained in great detail here.

As in most BASICs, both positive and negative STEPs are allowed in the FOR statement, and a STEP of +1 is assumed if the STEP portion of the statement is omitted. A FOR loop is terminated by a NEXT <variable> statement, and the <variable> must be the same as that referred to in the FOR statement at the beginning of the loop.

FOR loops may be nested four levels deep; NIBL reports an error if this limit is exceeded, or if a NEXT statement occurs without a previous FOR statement. As with the DO and UNTIL statements, FOR and NEXT may not be executed in edit mode.

Perhaps the only differences between the NIBL FOR statement and that of more elaborate BASICs (such as DEC's BASIC-PLUS for the PDP-11) are that a FOR loop is always executed at least once, and that when a NEXT statement is executed, the STEP value is added to the variable before the test is made to determine if the loop should be repeated (rather than after the test).

INPUT STATEMENT

There are two types of INPUT statements in NIBL: numeric input and string input. The form of the first type is 'INPUT' followed by a list of one or more variables. When this statement is executed, NIBL prompts at the teletype with a question mark ("?"). The user responds with a list of expressions separated by commas, and terminated by a carriage return. For example, a legal response to the statement INPUT A,B,C would be #3FA,26,4*27. These three expressions would then be assigned to the variables A, B, and C, respectively. An illegal response (too few arguments or improper expressions) will result in a syntax error. Any extra arguments in the response are ignored.

The second type of INPUT statement allows strings to be input. The form of the statement is 'INPUT' '$' <address>, where <address> is a Factor, syntactically (usually a variable, constant, or expression in parentheses). When this statement is executed, NIBL prompts the user as before, at which point the user enters a line terminated by the usual carriage return. NIBL then stores the line in memory in consecutive locations, beginning at the address specified. Thus, INPUT$ #6000 would cause the input line to be stored starting at location 6000 (base 16); the carriage return would also be stored at the end of the line.

Strings input in this manner can be tested and manipulated by using the "@" operator or the string handling statements described below. They can also be displayed by a PRINT statement.

Neither of the two INPUT statements may be executed in edit mode.

PRINT STATEMENT

The form of the PRINT statement is 'PRINT' or 'PR' followed by a list of print items separated by commas, and optionally terminated by a semicolon, which suppresses an otherwise automatic carriage return after all items in the list are printed.

A print item consists of one of the following:

1. A quoted string, which is printed exactly as it appears (with the quotes removed)
2. An expression, which is evaluated and printed in decimal format, with either a leading space or a minus sign ("-"), and one trailing space
3. A reference to a string in memory, denoted by '$' <address>, where <address> is a Factor as usual. Successive memory locations, starting at the specified address, are printed as ASCII characters, until a carriage return (which is not printed) is encountered.

There is no zone spacing in the PRINT statement, nor does NIBL perform an automatic carriage return/line feed after printing 72 characters. NIBL is not an output-oriented language; fancy formatting has been sacrificed for more useful control structures and data manipulation features. (A subroutine to print a number and skip to the next print zone is trivial to write in NIBL – it takes about two lines of code, with the DO/UNTIL and FOR/NEXT.)

STRING HANDLING STATEMENTS

String handling in NIBL is very minimal and low-level. The string handling features of the INPUT and PRINT statements have already been mentioned; NIBL provides two more statements for manipulating strings.

A statement such as $<address> = "THIS IS A STRING" would cause the quoted string to be stored in memory starting at the specified address (which again is a Factor), with a carriage return being appended to the string.

Another statement allows the programmer to move strings around in memory once they have been created. The form of this statement is '$' <destination> '=' '$' <source>, where both <destination> and <source> are Factors, and are the addresses of strings in memory. This statement causes all the characters in the string pointed to by <source> to be copied one-by-one to the memory pointed to by <destination>, until a carriage return (also copied) is encountered. Overlapping the source and destination addresses can produce disastrous results, such as wiping out the entire contents of the current page. Consequently, a string move can be aborted by hitting the BREAK key on the teletype (but it must be done quickly!).

Note that all strings referred to in these statements, and in the INPUT and PRINT statements, are assumed to lie within a 4K page, and wraparound is a possibility which must be anticipated by the programmer. (Long-time SC/MP programmers will be familiar with this minor problem.)

Using these statements, it should be very easy to develop a set of NIBL subroutines for performing concatenation, comparison, and substring operations on strings.

END STATEMENT

The END statement may appear anywhere in a NIBL program and not necessarily at the end. It causes a message and the current line number to be printed, with NIBL returning to edit mode. The END statement is useful when debugging programs, since it acts as a breakpoint in the program that can be removed easily.

LINK STATEMENT

The LINK statement allows NIBL programs to call SC/MP machine language routines at any address. A statement of the form 'LINK' <address>, where <address> is an arbitrary expression, will cause the NIBL system to call the routine at that address by executing an appropriate XPPC P3 instruction. The user's routine should make sure that it returns by executing another XPPC P3, and that the value of P3 upon entry to the routine is restored before returning. The routine may make use of the fact that P2 is set by NIBL to point to the beginning of the RAM block used to store the variables A through Z, with each variable being stored low byte first, high byte second. Thus, parameters may be passed between NIBL programs and machine language routines through the variables. Both P1 and P2 may be modified by the user's routines; they are automatically restored by the NIBL system upon return. The user should be careful not to modify RAM locations with negative displacements relative to P2, or the locations with displacements greater than 51 relative to P2. These locations are used by the interpreter.

REMARK STATEMENT

A comment can be inserted into a NIBL program by preceding it with the word REM. REM causes the rest of the line to be ignored by NIBL during execution. Remarks are useful in debugging programs or helping other people to understand them, but of course, they take up valuable memory. (Then again, memory is getting cheaper all the time.)

MULTIPLE STATEMENTS ON ONE LINE

A program line may contain more than one statement, if the statements are separated by colons (":"). Using multiple statements on a single line improves the readability of the program by separating it into small blocks, and uses less memroy for storing the program.

It is important to note that an IF statement will cause any statements appearing after it on the line to be ignored if the IF condition turns out to be false. This is the feature that allows a group of statements to be executed conditionally.

A multiple-statement line may be entered without a line number but NIBL will only execute the first statement on the line, ignoring the rest.

POWERING UP

NIBL is capable of executing a program in ROM in Page 2 immediately upon powering up, without the need for the user to give the RUN command at the teletype. When NIBL initializes, it examines Page 2 and makes an educated guess about the possible existence of a legal NIBL program in that page. If NIBL thinks there really is a program there, it starts executing it immediately; thus, if the program halts for some reason, the value of PAGE will be 2. But if NIBL fails to find a legal program in Page 2 initially, it sets the value of PAGE to 1 (the normal case) and prompts at the teletype.

When executing programs, NIBL periodically checks for keyboard interrupt, returning to edit mode if it detects it. Therefore, if a NIBL program is to be executed with the teletype disconnected, the Sense B line of the SC/MP should be set high so that NIBL will not sense an interrupt while running. This would allow a NIBL system to act as a process controller which starts executing immediately upon powering up.

BIOGRAPHICAL NOTE

Mark Alexander, a graduate of the University of California, Santa Cruz, is getting bored with assembly language programming, and wishes someone would save him by making a microprocessor copy of the Burroughs B5500 computer.

TABLE 1: NIBL Grammar

On reading the grammar:

All items in single quotes are actual symbols in NIBL; all other identities are symbols in the grammar. The equals sign "=", means "is defined as"; parentheses are used to group several items together as one item; the exclamation point, "!". means an exclusive-or choice between the items on either side of it; the asterisk, "*", means zero or more occurrences of the item to its left; the plus sign, "+", means one or more repetitions; the question mark, "?", means zero or one occurrences; and the semicolon, ";". marks the end of a definition.

```
NIBL-Line = Immediate-Statement
          ! Program-Line
          ;

Immediate-Statement = (Command ! Statement) Carriage-Return;

Program-Line = (Decimal-Number Statement-List Carriage-Return);

Command = 'NEW'
        ! 'CLEAR'
        ! 'LIST' Decimal-Number ?
        ! 'RUN'
        ;

Statement-List = Statement (':' STATEMENT) *;

Statement = 'LET' ? Left-part '=' Rel-Exp
          ! 'LET' ? '$' Factor '=' (String ! '$' Factor)
          ! 'GO' ('TO' ! 'SUB') Rel-Exp
          ! 'RETURN'
          ! ('PR' ! 'PRINT') Print-List
          ! 'IF' Rel-Expr 'THEN' ? Statement
          ! 'DO'
          ! 'UNTIL' Rel-Exp
          ! 'FOR' Variable '=' Rel-Exp 'TO' Rel-Exp ('STEP' Rel-Exp) ?
          ! 'NEXT' Variable
          ! 'INPUT' (Variable + ! '$' Factor)
          ! 'LINK' Rel-Exp
          ! 'REM' Any-Character-Except-Carriage-Return +
          ! 'END'
          ;

Left-Part = (Variable ! '@' Factor ! 'STAT' ! 'PAGE') ;

Rel-Exp = Expression Relop Expression
        ! Expression
        ;

Relop = '<' ! '<' '=' ! '<' '>' ! '>' ! '>' '=' ! '=' ;

Expression = Expression Adding-Operator term
           ! ('+' ! '-') ? Term
           ;

Adding-Operator = '+' ! '-' ! 'OR' ;

Term = Term Multiplying-Operator Factor
     ! Factor
     ;

Multiplying-Operator = '*' ! '/' ! 'AND' ;

Factor = Variable
       ! Decimal-Number
       ! '(' Rel-Exp ')'
       ! '@' Factor
       ! '$' Hex-Number
       ! 'NOT' Factor
       ! 'MOD' '(' Rel-Exp ',' Rel-Exp ')'
       ! 'RND' '(' Rel-Exp ',' Rel-Exp ')'
       ! 'STAT'
       ! 'TOP'
       ! 'PAGE'
       ;

Variable = 'A' ! 'B' ! 'C' ! ... ! 'Y' ! 'Z' ;

Decimal-Number = Decimal-Digit + ;

Decimal-Digit = '0' ! '1' ! '2' ! ... ! '9' ;

Hex-Number = (Decimal-Digit ! Hex-Digit) + ;

Hex-Digit = 'A' ! 'B' ! 'C' ! 'D' ! 'E' ! 'F' ;

Print-list = Print-Item + ;

Print-Item = (String ! Rel-exp ! '$' Factor) ;

String = '"' Almost-Any-Character '"' ;
```

NOTE: Spaces are not usually significant in NIBL programs, with the following exceptions: spaces cannot appear within key words (such as 'THEN' or 'UNTIL') or within constants. Also, a variable (such as A or Z) must be followed immediately by a non-alphabetic character to distinguish it from a key word.

TABLE 1: NIBL Grammar

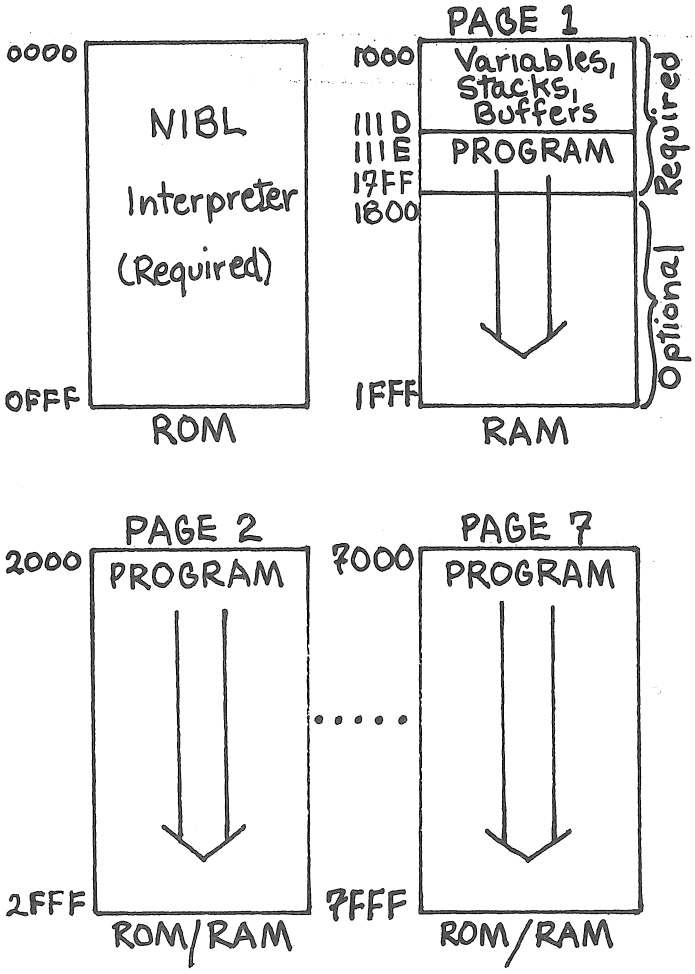
TABLE 2: NIBL error messages

Error messages are of the form:

EEEE ERROR AT LN

where EEEE is one of the error codes below, and LN is the number of of the line in which the error was encountered.

AREA	No more room left for program in current page
CHAR	Character after logical end of statement
DIV0	Division by zero
END"	No ending quote on string
FOR	FOR without NEXT
NEST	Nesting limit exceeded in expression, FOR's, GOSUBs, etc.
NEXT	NEXT without FOR
NOGO	No line number corresponding to GOTO or GOSUB
RTRN	RETURN without previous GOSUB
SNTX	Syntax error
STMT	Statement type used improperly
UNTL	UNTL without DO
VALU	Constant format or value error



CODE FOLLOWS

KILOBAUD – A PRENATEL NAME CHANGE

John Craig, the Editor of Wayne Green's new computer hobby mag, just phoned and told us that Wayne has changed the publication's name – before the first issues comes out – from the initially advertised *"Kilobyte"* to *Kilobaud.* Oh, well . . . we're still waiting for someone to start yet another rag and call it *"Megabyte"* (but with luck, that won't happen).

COMPUTER HOBBYIST CONVENTIONS & TRADE SHOWS

CONVENTIONS ALREADY HELD:

May 2, 1976	Trenton Festival Trenton, NJ Amateur Comp. Group of NJ	1500 people 45 exhibitors
June 11-13, 1976	Midwest Reg. Comp. Conf. Cleveland, OH Midwest Affiliation of Comp. Clubs	1500-2500 people
Aug 28-29, 1976	Personal Computing '76 Atlantic City, NJ S. Counties Amateur Radio Assn. of NJ	4500-5000 people; 103 exhibitors

CONVENTIONS BELIEVED TO BE IN THE WORKS:

Mar 5, 1977 (Saturday) 10 AM - 3 PM	Microprocessor Hobbyists Demo United Good Neighbor Bldg. Renton, WA (Not a convention, but interesting)	Mike & Key Amateur Radio Club Bill Balzarini K7MWC 1518 S. Pearl St. Seattle, WA 98108 (206) 762-7738
Mar 19-20, 1977	Western Personal Computing Show Hyatt House, International Airpt. Los Angeles	Austin Cragg Conference & Exposition Management Co., Box 844 Greenwich, CT 06830 (203) 661-6101
Apr 15-17, 1977	**The First West Coast Computer Faire, Civic Auditorium San Francisco, CA San Francisco, CA [Expecting 7,000-10,000 people, 50 sessions, 200 exhibitors]**	**[co-sponsored by a number of Bay Area hobbyist, professional and educational organizations]**
Apr 31-May 1, 1977	Trenton Computerfest Trenton, NJ	Alan Katz Dept. of Engr., Trenton State Coll., Trenton, NJ 08625 (609) 771-2487
May 7-8, 1977	Eastern Personal Computing Show, Mariott Hotel Philadelphia, PA	Austin Cragg [listed prev.]
Jun 13-16, 1977	**Personal Computing Section National Computer Conference '77 Dallas, TX**	**AFIPS 210 Summit Ave. Montvale, NJ 07645 (201) 391-9810**
Jun 18-19, 1977	New England Personal Comp. Show, J.B. Hynes Aud. Boston, MA	Austin Cragg [listed prev.]
Jun 18-19, 1977	Atlanta Computerfest Atlanta, GA [in conjunction with Hamfest]	? *'73 Magazine* 73 Pine St. Peterborough, NH 03458 (603) 924-3873
Jun, 1977	Midwest Reg. Comp. Conf. Cleveland, OH	Midwest Affiliation of Comp. Clubs, PO Box 83 Brecksville, OH 44141 (216) 732-8458
Jul 29-31, 1977	Northwestern Amateur Radio Convention Seattle Ctr. & Washington Plaza Hotel, Seattle, WA [will include significant microcomputer activities]	ARRL-QCWA-WWDX Club ARRL Conven. Comm. 10352 Sand Point Way NE Seattle, WA 98125
Aug 27-28, 1977	Personal Computing '77 Consumer Trade Fair Atlantic City, NJ [?]	John Dilks, PC'77 503 W. New Jersey Ave. Somers Pt., NJ 08244 (609) 927-6950
Oct 25-28, 1977	(Name unknown at press time) Anaheim Conv. Ctr. Anaheim, CA	*Interface Age* Box 1234 Cerritos, CA 90701 (213) 469-7789
Fall, 1977	(Name unknown at press time) Los Angeles Area [Proposal to hold such a convention has been placed before SCCS Bd. of Directors]	Southern California Computer Society P.O. Box 3123 Los Angeles, CA 90051
???	Technihobby-USA [3 of the 4 listed previously were postponed. Last word was they were considering also postponing the 4th.]	Marketing Ventures, Inc. 5012 Herzel Pl. Beltsville, MD 20705 (301) 937-7177

Note: This list excludes a number of conventions directed towards computer professionals that are expected to have at least nominal activity in the area of personal and hobby computing. Although the '77 NCC is primarily for computer professionals, its Personal Computing Section will be a major activity with a number of significant sessions and events planned for personal computer enthusiasts.

```
                .TITLE  NIBL,'NOV. 27'
0001            .LIST   1

        ;*******************************************************
        ;*                                                     *
        ;*      WE ARE TIED DOWN TO A LANGUAGE WHICH           *
        ;*      MAKES UP IN OBSCURITY WHAT IT LACKS            *
        ;*      IN STYLE.                                      *
        ;*                      -- TOM STOPPARD                *
        ;*                                                     *
        ;*******************************************************

0020    TSTBIT  =       020             ;I.L. INSTRUCTION FLAGS
0040    JMPBIT  =       040
0080    CALBIT  =       080
0001    P1      =       1               ;SC/MP POINTER ASSIGNMENTS
0002    P2      =       2
0003    P3      =       3
FF80    EREG    =       -128            ;THE EXTENSION REGISTER

        ; DISPLACEMENTS FOR RAM VARIABLES USED BY INTERPRETER

FFFF    DOPTR   =       -1              ;DO-STACK POINTER
FFFE    FORPTR  =       -2              ;FOR-STACK POINTER
FFFD    LSTK    =       -3              ;ARITHMETIC STACK POINTER
FFFC    SBRPTR  =       -4              ;GOSUB STACK POINTER
FFFB    PCLOW   =       -5              ;I.L. PROGRAM COUNTER
FFFA    PCHIGH  =       -6
FFF9    PCSTK   =       -7              ;I.L. CALL STACK POINTER
FFF8    LOLINE  =       -8              ;CURRENT LINE NUMBER
FFF7    HILINE  =       -9
FFF6    PAGE    =       -10             ;VALUE OF CURRENT PAGE
FFF5    LISTNG  =       -11             ;LISTING FLAG
FFF4    RUNMOD  =       -12             ;RUN/EDIT FLAG
FFF3    LABLLO  =       -13
FFF2    LABLHI  =       -14
FFF1    P1LOW   =       -15             ;SPACE TO SAVE CURSOR
FFF0    P1HIGH  =       -16
FFEF    LO      =       -17
FFEE    HI      =       -18
FFED    FAILLO  =       -19
FFEC    FAILHI  =       -20
FFEB    NUM     =       -21
FFEA    TEMP    =       -22
FFE9    TEMP2   =       -23
FFE8    TEMP3   =       -24
FFE7    CHRNUM  =       -25
FFE6    RNDF    =       -26
FFE5    RNDX    =       -27             ;SEEDS FOR RANDOM NUMBER
FFE4    RNDY    =       -28

        ; ALLOCATION OF RAM FOR NIBL VARIABLES, STACKS,
        ;  AND LINE BUFFER
0000            .=01000+28
101C    VARS:   .=.+52                  ; NIBL VARIABLES A-Z
1050    AESTK:  .=.+26                  ; ARITHMETIC STACK
106A    SBRSTK: .=.+16                  ;GOSUB STACK
107A    DOSTAK: .=.+16                  ;DO/UNTIL STACK
108A    FORSTK: .=.+28                  ;FOR/NEXT STACK
10A6    PCSTAK: .=.+48                  ;I.L. CALL STACK
10D6    LBUF:   .=.+74                  ;LINE BUFFER
1120    PGM:    .=0                     ;USER'S PROGRAM

                .MACRO  LDPI,P,VAL
                .MLOC   TEMP
                .SET    TEMP,VAL
                LDI     H(TEMP)
                XPAH    P
                LDI     L(TEMP)
                XPAL    P
                .ENDM


        ;*************************************
        ;*      INITIALIZATION OF NIBL       *
        ;*************************************

0000 08                 NOP
0001                    LDPI    P2,VARS         ;POINT P2 AT VARIABLES
0007                    LDPI    P1,PGM          ;POINT P1 AT PAGE ONE PROGRAM
000D C4FF               LDI     -1              ;STORE -1 AT START OF PROGRAM
000F C900               ST      0(P1)
0011 C901               ST      1(P1)
0013 C40D               LDI     0D              ;ALSO STORE A DUMMY END-OF-LI
0015 C9FF               ST      -1(P1)
0017 C402               LDI     2               ;POINT P2 AT PAGE 2,
0019 CAF6               ST      PAGE(P2)        ; INITIALLY SET PAGE TO 2
001B 31                 XPAL    P1
001C C420               LDI     020
001E 35                 XPAH    P1
001F B902               DLD     2(P1)           ;CHECK IF THERE IS REALLY
0021 01                 XAE                     ; A PROGRAM IN PAGE 2:
0022 C180               LD      EREG(P1)        ; IF FIRST LINE LENGTH
0024 E40D               XRI     0D              ; POINTS TO CARR. RETURN
0026 9802               JZ      $0              ; AT END OF LINE
0028 BAF6               DLD     PAGE(P2)        ;IF NOT, PAGE = 1
002A C420       $0:     LDI     020
002C 35         $LOOP:  XPAH    P1
002D C4FF               LDI     -1              ;STORE -1 IN 2 CONSECUTIVE
002F C900               ST      (P1)            ; LOCATIONS AT START OF PAGE
0031 C901               ST      1(P1)
0033 C40D               LDI     0D              ;ALSO PUT A DUMMY END-OF-LINE
0035 C9FF               ST      -1(P1)          ; JUST BEFORE TEXT
0037 35                 XPAH    P1              ;UPDATE P1 TO POINT TO
0038 02                 CCL                     ; NEXT PAGE (UNTIL PAGE=8)
0039 F410               ADI     010             ;REPEAT INITIALIZATION
003B E480               XRI     080             ; FOR PAGES 2-7
003D 9804               JZ      $1
003F E480               XRI     080
0041 90E9               JMP     $LOOP
0043 C400       $1:     LDI     0               ;CLEAR SOME FLAGS
0045 CAF4               ST      RUNMOD(P2)
0047 CAF5               ST      LISTNG(P2)
0049 C454               LDI     L(BEGIN)        ;INITIALIZE IL PC SO THAT
004B CAFB               ST      PCLOW(P2)       ; NIBL PROGRAM
004D C40C               LDI     H(BEGIN)        ; IS EXECUTED IMMEDIATELY
004F CAFA               ST      PCHIGH(P2)
0051 C400       CLEAR:  LDI     0
0053 CAEA               ST      TEMP(P2)
0055 01                 XAE
0056 C400       CLEAR1: LDI     0               ;SET ALL VARIABLES
0058 CA80               ST      EREG(P2)        ; TO ZERO
005A AAEA               ILD     TEMP(P2)
005C 01                 XAE
005D C434               LDI
005F 60                 XRE
0060 9CF4               JNZ     CLEAR1
0062 C450               LDI     L(AESTK)        ;INITIALIZE SOME STACKS:
0064 CAFD               ST      LSTK(P2)        ; ARITHMETIC STACK,
0066 C47A               LDI     L(DOSTAK)
0068 CAFF               ST      DOPTR(P2)       ; DO/UNTIL STACK,
006A C46A               LDI     L(SBRSTK)
006C CAFC               ST      SBRPTR(P2)      ; GOSUB STACK,
006E C4A6               LDI     L(PCSTAK)
0070 CAF9               ST      PCSTK(P2)       ; I.L. CALL STACK,
0072 C48A               LDI     L(FORSTK)
0074 CAFE               ST      FORPTR(P2)      ; FOR/NEXT STACK


        ;*************************************
        ;*   INTERMEDIATE LANGUAGE EXECUTOR  *
        ;*************************************

0076 C2FB       EXECIL: LD      PCLOW(P2)       ;SET P3 TO CURRENT
0078 33                 XPAL    P3              ; IL PC.
0079 C2FA               LD      PCHIGH(P2)
007B 37                 XPAH    P3
007C C701       CHEAT:  LD      @1(P3)
007E 01                 XAE                     ;GET NEW IL INSTRUCTION
007F C701               LD      @1(P3)          ; INTO P3 THROUGH
0081 33                 XPAL    P3              ; OBSCURE METHODS
0082 CAFB               ST      PCLOW(P2)       ;SIMULTANEOUSLY, INCREMENT
0084 40                 LDE                     ; THE IL PC BY 2
0085 37                 XPAH    P3
0086 CAFA               ST      PCHIGH(P2)
0088 40                 LDE
0089 D4F0               ANI     0F0             ;CHECK IF IL INSTRUCTION
008B E420               XRI     TSTBIT          ; IS A 'TEST'
008D 9836               JZ      TST
008F E4A0               XRI     CALBIT!TSTBIT   ;CHECK FOR IL CALL
0091 980D               JZ      ILCALL
0093 E4C0               XRI     JMPBIT!CALBIT   ;CHECK FOR IL JUMP
0095 9C06               JNZ     NOJUMP
0097 37                 XPAH    P3              ;*** I.L. JUMP ***
0098 D40F               ANI     0F              ;ALL IT TAKES IS SCRUBBING
009A 37                 XPAH    P3              ; THE JUMP FLAG OFF OF P3
009B 90DF       CHEAT1: JMP     CHEAT
009D 3F         NOJUMP: XPPC    P3              ;MUST BE AN ML SUBROUTINE
009E 90D6               JMP     EXECIL          ; IF NONE OF THE ABOVE


        ;*************************************
        ;*     INTERMEDIATE LANGUAGE CALL    *
        ;*************************************

00A0 C2F9       ILCALL: LD      PCSTK(P2)
00A2 E4D6               XRI     L(LBUF)         ;CHECK FOR STACK OVERFLOW
00A4 9C04               JNZ     ILC1
00A6 C40A               LDI     10
00A8 9063               JMP     EOA
00AA E4D6       ILC1:   XRI     L(LBUF)         ;RESTORE ACCUMULATOR
00AC 33                 XPAL    P3              ;SAVE LOW BYTE OF NEW
00AD CAEA               ST      TEMP(P2)        ; I.L. PC IN TEMP
00AF C410               LDI     H(PCSTAK)       ;POINT P3 AT I.L.
00B1 37                 XPAH    P3              ; SUBROUTINE STACK
00B2 C2FB               LD      PCLOW(P2)       ;SAVE OLD I.L. PC ON STACK
00B4 CF01               ST      @1(P3)
00B6 C2FA               LD      PCHIGH(P2)
00B8 CF01               ST      @1(P3)
00BA C2EA               LD      TEMP(P2)        ;GET LOW BYTE OF NEW
00BC 33                 XPAL    P3              ; I.L. PC INTO P3 LOW
00BD CAF9               ST      PCSTK(P2)       ;UPDATE I.L. STACK POINTER
00BF 40                 LDE                     ;GET HIGH BYTE OF NEW P3
00C0 D40F               ANI     0F              ; I.L. PC INTO P3 HIGH
00C2 37                 XPAH    P3
00C3 90B7               JMP     CHEAT


        ;*************************************
        ;*      I.L. 'TEST' INSTRUCTION      *
        ;*************************************

                .LOCAL
00C5 CAE7       TST:    ST      CHRNUM(P2)      ;CLEAR NUMBER OF CHARS SCANNE
00C7 C501       $SCAN:  LD      @1(P1)          ;SLEW OFF SPACES
00C9 E420               XRI     ' '
00CB 98FA               JZ      $SCAN
00CD C5FF               LD      @-1(P1)         ;REPOSITION CURSOR
00CF C2FA               LD      PCHIGH(P2)      ;POINT P3 AT IL TABLE
00D1 37                 XPAH    P3
00D2 D40F               ANI     0F              ;FAIL ADDRESS <- OLD P3
00D4 CAEC               ST      FAILHI(P2)
00D6 C2FB               LD      PCLOW(P2)
00D8 33                 XPAL    P3
00D9 CAED               ST      FAILLO(P2)
00DB C701       $LOOP:  LD      @1(P3)
00DD 01                 XAE                     ;SAVE CHAR FROM TABLE
00DE BAE7               DLD     CHRNUM(P2)      ;DECREMENT CHAR COUNT
00E0 40                 LDE                     ;GET CHAR BACK
00E1 D47F               ANI     07F             ;SCRUB OFF FLAG (IF ANY)
00E3 E501               XOR     @1(P1)          ;IS CHAR EQUAL TO TEXT CHAR?
00E5 9C07               JNZ     $NEQ            ;NO - END TEST
00E7 40                 LDE                     ;YES - BUT IS IT LAST CHAR?
00E8 94F1               JP      $LOOP           ;IF NOT, CONTINUE TO COMPARE
00EA 9090               JMP     CHEAT           ;IF SO, GET NEXT I.L.
00EC 9088       XO:     JMP     EXECIL          ; INSTRUCTION
00EE C2E7       $NEQ:   LD      CHRNUM(P2)      ;RESTORE P1 TO
00F0 01                 XAE                     ; ORIGINAL VALUE
00F1 C580               LD      @EREG(P1)
00F3 C2ED               LD      FAILLO(P2)      ;LOAD TEST-FAIL ADDRESS
00F5 33                 XPAL    P3              ; INTO P3
00F6 C2EC               LD      FAILHI(P2)
00F8 37                 XPAH    P3
00F9 90A0               JMP     CHEAT1          ;GET NEXT IL INSTRUCTION


        ;*************************************
        ;*       I.L. SUBROUTINE RETURN      *
        ;*************************************
```

```
00FB C410  RTN:    LDI     H(PCSTAK)       ;POINT P3 AT I.L. PC STACK
00FD 37            XPAH    P3
00FE C2F9          LD      PCSTK(P2)
0100 33            XPAL    P3
0101 C7FF          LD      @-1(P3)         ;GET HIGH PART OF OLD PC
0103 01            XAE
0104 C7FF          LD      @-1(P3)         ;GET LOW PART OF OLD PC
0106 33            XPAL    P3
0107 CAF9          ST      PCSTK(P2)       ;UPDATE IL STACK POINTER
0109 40            LDE
010A 37            XPAH    P3              ;P3 NOW HAS OLD IL PC
010B 908E          JMP     CHEAT1
010D 9041  E0A:    JMP     E0

           ;************************************
           ;*     SAVE GOSUB RETURN ADDRESS    *
           ;************************************

010F C2FC  SAV:    LD      SBRPTR(P2)
0111 E47A          XRI     L(DOSTAK)       ;CHECK FOR MORE
0113 981C          JZ      SAV2            ; THAN 8 SAVES
0115 AAFC          ILD     SBRPTR(P2)
0117 AAFC          ILD     SBRPTR(P2)
0119 33            XPAL    P3              ;SET P3 TO
011A C410          LDI     H(SBRSTK)       ; SUBROUTINE STACK TOP.
011C 37            XPAH    P3
011D C2F4          LD      RUNMOD(P2)      ;IF IMMEDIATE MODE,
011F 980A          JZ      SAV1            ; SAVE NEGATIVE ADDRESS.
0121 35            XPAH    P1              ;SAVE HIGH PORTION
0122 CBFF          ST      -1(P3)          ; OF CURSOR
0124 35            XPAH    P1
0125 31            XPAL    P1              ;SAVE LOW PORTION
0126 CBFE          ST      -2(P3)          ; OF CURSOR
0128 31            XPAL    P1
0129 90C1          JMP     X0              ;RETURN
012B C4FF  SAV1:   LDI     -1              ;IMMEDIATE MODE
012D CBFF          ST      -1(P3)          ; RETURN ADDRESS IS
012F 90BB          JMP     X0              ; NEGATIVE.
0131 C40A  SAV2:   LDI     10              ;ERROR: MORE THAN
0133 901B          JMP     E0              ; 8 GOSUBS

           ;************************************
           ;*     CHECK STATEMENT FINISHED     *
           ;************************************

0135 C501  DONE:   LD      @1(P1)          ;SKIP SPACES
0137 E420          XRI     ' '
0139 98FA          JZ      DONE
013B E42D          XRI     ' ' ! 0D        ;IS IT CARRIAGE RETURN?
013D 9804          JZ      DONE1           ;YES - RETURN
013F E437          XRI     037             ;IS CHAR A ':' ?
0141 9C01          JNZ     DONE2           ;NO - ERROR
0143 3F    DONE1:  XPPC    P3              ;YES - RETURN
0144 C404  DONE2:  LDI     4
0146 9008          JMP     E0

           ;************************************
           ;*     RETURN FROM GOSUB            *
           ;************************************

0148 C2FC  RSTR:   LD      SBRPTR(P2)
014A E46A          XRI     L(SBRSTK)       ;CHECK FOR RETURN
014C 9C04          JNZ     RSTR1           ; W/O GOSUB.
014E C409          LDI     9
0150 9043  E0:     JMP     E1              ;GOTO ERROR.
0152 BAFC  RSTR1:  DLD     SBRPTR(P2)
0154 BAFC          DLD     SBRPTR(P2)      ;POP GOSUB STACK,
0156 33            XPAL    P3              ; PUT PTR INTO P3.
0157 C410          LDI     H(SBRSTK)
0159 37            XPAH    P3
015A C301          LD      1(P3)           ;IF ADDRESS NEGATIVE,
015C 9409          JP      RSTR2           ; SUBROUTINE WAS CALLED
015E C402          JS      P3,FIN          ; IN IMMEDIATE MODE,
0165 9085  X1:     JMP     X0              ; SO FINISH UP EXECUTING
0167 35    RSTR2:  XPAH    P1              ;RESTORE CURSOR HIGH
0168 C300          LD      0(P3)
016A 31            XPAL    P1              ;RESTORE CURSOR LOW
016B C401          LDI     1               ;SET RUN MODE
016D CAF4          ST      RUNMOD(P2)
016F 90F4          JMP     X1

           ;************************************
           ;*     TRANSFER TO NEW STATEMENT    *
           ;************************************

0171 C2F2  XFER:   LD      LABLHI(P2)      ;CHECK FOR NON-EXISTENT LINE
0173 9404          JP      XFER1
0175 C408          LDI     8
0177 901C          JMP     E1
0179 C401  XFER1:  LDI     1               ;SET RUN MODE TO 1
017B CAF4          ST      RUNMOD(P2)
017D 3F            XPPC    P3

           ;************************************
           ;*     PRINT STRING IN TEXT         *
           ;************************************

017E       PRS:    LDPI    P3,PUTC-1       ;POINT P3 AT PUTC ROUTINE
0184 C501          LD      @1(P1)          ;LOAD NEXT CHAR
0186 E422          XRI     '"'             ;IF ", END OF
0188 98DB          JZ      X1              ; STRING
018A E42F          XRI     02F             ;IF CR, ERROR
018C 9805          JZ      PRS1
018E E40D          XRI     0D              ;RESTORE CHAR
0190 3F            XPPC    P3              ;PRINT CHAR
0191 90EB          JMP     PRS             ;GET NEXT CHAR
0193 C407  PRS1:   LDI     7               ;SYNTAX ERROR
0195 9035  E1:     JMP     E2

           ;************************************
           ;*     PRINT NUMBER ON STACK        *
           ;************************************

           ; THIS ROUTINE IS BASED ON DENNIS ALLISON'S BINARY TO DECIMAL
           ; CONVERSION ROUTINE IN VOL. 1, #1 OF "DR. DOBB'S JOURNAL",
           ; BUT IS MUCH MORE OBSCURE BECAUSE OF THE STACK MANIPULATION.

                      .LOCAL
0197 C410  PRN:    LDI     H(AESTK)        ;POINT P3 AT A.E. STACK
0199 37            XPAH    P3
019A AAFD          ILD     LSTK(P2)
019C AAFD          ILD     LSTK(P2)
019E 33            XPAL    P3
019F C40A          LDI     10              ;PUT 10 ON STACK (WE'LL BE
01A1 CBFE          ST      -2(P3)          ; DIVIDING BY IT LATER)
01A3 C400          LDI     0
01A5 CBFF          ST      -1(P3)
01A7 C405          LDI     5               ;SET CHRNUM TO POINT TO PLACE
01A9 CAE7          ST      CHRNUM(P2)      ; IN STACK WHERE WE STORE
01AB C4FF          LDI     -1              ; THE CHARACTERS TO PRINT
01AD CB05          ST      5(P3)           ;FIRST CHAR IS A FLAG (-1)
01AF C3FD          LD      -3(P3)          ;CHECK IF NUMBER IS NEGATIVE
01B1 9413          JP      $1
01B3 C42D          LDI     '-'             ;PUT '-' ON STACK, AND NEGATE
01B5 CB04          ST      4(P3)           ; THE NUMBER
01B7 C400          LDI     0
01B9 03            SCL
01BA FBFC          CAD     -4(P3)
01BC CBFC          ST      -4(P3)
01BE C400          LDI     0
01C0 FBFD          CAD     -3(P3)
01C2 CBFD          ST      -3(P3)
01C4 909F          JMP     X1              ;GO DO DIVISION BY 10
01C6 C420  $1:     LDI     ' '             ;IF POSITIVE, PUT ' ' ON
01C8 CB04          ST      4(P3)           ; STACK BEFORE DIVISION
01CA 9099  X4:     JMP     X1
01CC 9057  E2:     JMP     ERR1

           ; THE DIVISION IS PERFORMED, THEN CONTROL IS TRANSFERRED
           ; TO PRN1, WHICH FOLLOWS.

01CE AAFD  PRN1:   ILD     LSTK(P2)        ;POINT P1 AT A.E. STACK
01D0 AAFD          ILD     LSTK(P2)
01D2 31            XPAL    P1
01D3 C410          LDI     H(AESTK)
01D5 35            XPAH    P1
01D6 AAE7          ILD     CHRNUM(P2)      ;INCREMENT CHARACTER STACK
01D8 01            XAE                     ; POINTER, PUT IN EX. REG.
01D9 C101          LD      1(P1)           ;GET REMAINDER FROM DIVIDE,
01DB DC30          ORI     '0'
01DD C980          ST      EREG(P1)        ; PUT IT ON THE STACK
01DF C1FD          LD      -3(P1)          ;IS THE QUOTIENT ZERO YET?
01E1 D9FC          OR      -4(P1)
01E3 980A          JZ      $PRNT           ;YES - GO PRINT THE NUMBER
01E5 C40F          LDI     H(PRNUM1)       ;NO - CHANGE THE I.L. PC
01E7 CAFA          ST      PCHIGH(P2)      ; SO THAT DIVIDE IS
01E9 C42F          LDI     L(PRNUM1)       ; PERFORMED AGAIN
01EB CAFB          ST      PCLOW(P2)
01ED 90DB          JMP     X4              ;GO DO DIVISION BY 10 AGAIN
01EF       $PRNT:  LDPI    P3,PUTC-1       ;POINT P3 AT PUTC ROUTINE
01F5 C2F5          LD      LISTNG(P2)      ;IF LISTING, SKIP PRINTING
01F7 9C06          JNZ     $2              ; LEADING SPACE
01F9 C104          LD      4(P1)           ;PRINT EITHER '-'
01FB 3F            XPPC    P3              ; OR LEADING SPACE
01FC C2E7          LD      CHRNUM(P2)      ;GET EX. REG. VALUE BACK
01FE 01            XAE
01FF C580  $2:     LD      @EREG(P1)       ;POINT P3 AT FIRST CHAR
0201 C100          LD      (P1)            ; TO BE PRINTED
0203 3F    $LOOP:  XPPC    P3              ;PRINT THE CHARACTER
0204 C5FF          LD      @-1(P1)         ;GET NEXT CHARACTER
0206 94FB          JP      $LOOP           ;REPEAT UNTIL = -1
0208 C450          LDI     L(AESTK)
020A CAFD          ST      LSTK(P2)        ;CLEAR THE A.E. STACK
020C C2F5          LD      LISTNG(P2)      ;PRINT A TRAILING SPACE
020E 9CBA          JNZ     X4              ; IF NOT LISTING PROGRAM
0210 C420          LDI     ' '
0212 3F            XPPC    P3
0213 90B5          JMP     X4

           ;************************************
           ;*     CARRIAGE RETURN/LINE FEED    *
           ;************************************

0215       NLINE:  LDPI    P3,PUTC-1       ;POINT P3 AT PUTC ROUTINE
021B C40D          LDI     0D              ;CARRIAGE RETURN
021D 3F            XPPC    P3
021E C40A          LDI     0A              ;LINE FEED
0220 3F            XPPC    P3
0221 90A7  X5:     JMP     X4

           ;************************************
           ;*     ERROR  ROUTINE               *
           ;************************************

                      .LOCAL
0223 C405  ERR:    LDI     5               ;SYNTAX ERROR
0225 CAEB  ERR1:   ST      NUM(P2)         ;SAVE ERROR #
0227 C2EB  ERR2:   LD      NUM(P2)
0229 CAEA          ST      TEMP(P2)
022B               LDPI    P3,PUTC-1       ;POINT P3 AT PUTC
0231 C40D          LDI     0D              ;PRINT CR/LF
0233 3F            XPPC    P3
0234 C40A          LDI     0A
0236 3F            XPPC    P3
0237               LDPI    P1,MESGS        ;P1 -> ERROR MESSAGES
023D BAEB  $1:     DLD     NUM(P2)         ;IS THIS THE RIGHT MESSAGE?
023F 9806          JZ      $MSG            ;YES - GO PRINT IT
0241 C501  $LOOP:  LD      @1(P1)          ;NO - SCAN THROUGH TO
0243 94FC          JP      $LOOP           ; NEXT MESSAGE
0245 90F6          JMP     $1
0247 C501  $MSG:   LD      @1(P1)          ;GET MESSAGE CHAR
0249 3F            XPPC    P3              ;PRINT IT
024A C1FF          LD      -1(P1)          ;IS MESSAGE DONE?
024C 94F9          JP      $MSG            ;NO - GET NEXT CHAR
024E C2EA          LD      TEMP(P2)        ;WAS THIS A BREAK MESSAGE?
0250 E40E          XRI     14
0252 980D          JZ      $3              ;YES - SKIP PRINTING 'ERROR'
0254               LDPI    P1,MESGS        ;NO - PRINT 'ERROR'
025A C501  $2:     LD      @1(P1)          ;GET CHARACTER
025C 3F            XPPC    P3              ;PRINT IT
025D C1FF          LD      -1(P1)          ;DONE?
```

```
025F 94F9            JP      $2              ;NO - REPEAT LOOP
0261 C2F4   $3:      LD      RUNMOD(P2)      ;DON'T PRINT LINE #
0263 984D            JZ      FIN             ; IF IMMEDIATE MODE
0265 C420            LDI     ' '
0267 3F              XPPC    P3              ;SPACE
0268 C441            LDI     'A'             ;AT
026A 3F              XPPC    P3
026B C454            LDI     'T'
026D 3F              XPPC    P3
026E C410            LDI     H(AESTK)        ;POINT P3 AT A.E. STACK
0270 37              XPAH    P3
0271 AAFD            ILD     LSTK(P2)
0273 AAFD            ILD     LSTK(P2)
0275 33              XPAL    P3
0276 C2F7            LD      HILINE(P2)      ;GET HIGH BYTE OF LINE #
0278 CBFF            ST      -1(P3)          ;PUT ON STACK
027A C2F8            LD      LOLINE(P2)      ;GET LOW BYTE OF LINE #
027C CBFE            ST      -2(P3)          ;PUT ON STACK
027E C42D            LDI     L(ERRNUM)       ;GO TO PRN
0280 CAFB            ST      PCLOW(P2)
0282 C40E            LDI     H(ERRNUM)
0284 CAFA            ST      PCHIGH(P2)
0286 9099   X5A:     JMP     X5

            ;************************************
            ;*      BREAK, NXT, FIN, & STRT     *
            ;************************************

0288 C40E   BREAK:   LDI     14
028A 9099   E3A:     JMP     ERR1
                                             ;*** NEXT STATEMENT ***
028C C2F4   NXT:     LD      RUNMOD(P2)      ;IF IN IMMED. MODE,
028E 9822            JZ      FIN             ; STOP EXECUTION
0290 C100            LD      (P1)            ;IF WE HIT END OF FILE,
0292 D480            ANI     080             ; FINISH UP THINGS
0294 9C1C            JNZ     FIN
0296 06              CSA                     ;BREAK IF SOMEONE IS
0297 D420            ANI     020             ; TYPING ON THE CONSOLE
0299 98ED            JZ      BREAK
029B C1FF            LD      -1(P1)          ;GET LAST CHARACTER SCANNED
029D E40D            XRI     0D              ;WAS IT CARRIAGE RETURN?
029F 9C08            JNZ     NXT1            ;YES - SKIP FOLLOWING UPDATES
02A1 C501            LD      @1(P1)          ;GET HIGH BYTE OF NEXT LINE #
02A3 CAF7            ST      HILINE(P2)      ;SAVE IT
02A5 C502            LD      @2(P1)          ;GET LOW BYTE OF LINE #, SKIP
02A7 CAF8            ST      LOLINE(P2)      ; LINE LENGTH BYTE
02A9 C40C   NXT1:    LDI     H(STMT)         ;GO TO 'STMT' IN IL TABLE
02AB CAFA            ST      PCHIGH(P2)
02AD C482            LDI     L(STMT)
02AF CAFB            ST      PCLOW(P2)
02B1 3F              XPPC    P3

02B2 C400   FIN:     LDI     0               ;*** FINISH EXECUTION ***
02B4 CAF4            ST      RUNMOD(P2)      ;CLEAR RUN MODE
02B6 C450            LDI     L(AESTK)        ;CLEAR ARITHMETIC STACK
02B8 CAFD            ST      LSTK(P2)
02BA C418            LDI     L(START)        ;SET IL PC TO GETTING LINES
02BC CAFB            ST      PCLOW(P2)
02BE C40C            LDI     H(START)
02C0 CAFA            ST      PCHIGH(P2)
02C2 C4A6            LDI     L(PCSTAK)
02C4 CAF9            ST      PCSTK(P2)
02C6 90BE            JMP     X5A
                                             ;*** START EXECUTION ***
02C8 AAF4   STRT:    ILD     RUNMOD(P2)      ;RUN MODE = 1
02CA C2E9            LD      TEMP2(P2)       ;POINT CURSOR TO
02CC 35              XPAH    P1              ; START OF NIBL PROGRAM
02CD C2E8            LD      TEMP3(P2)
02CF 31              XPAL    P1
02D0 C46A            LDI     L(SBRSTK)       ;EMPTY SOME STACKS:
02D2 CAFC            ST      SBRPTR(P2)      ; GOSUB STACK,
02D4 C48A            LDI     L(FORSTK)
02D6 CAFE            ST      FORPTR(P2)      ; FOR STACK
02D8 C47A            LDI     L(DOSTAK)
02DA CAFF            ST      DOPTR(P2)       ; & DO/UNTIL STACK
02DC 3F              XPPC    P3              ;RETURN
02DD 90A7   X6:      JMP     X5A
02DF 90A9   E4:      JMP     E3A

            ;************************************
            ;*        LIST NIBL PROGRAM         *
            ;************************************

02E1 C100   LST:     LD      (P1)            ;CHECK FOR END OF FILE
02E3 E480            XRI     080
02E5 9418            JP      LST2
02E7 C410            LDI     H(AESTK)        ;GET LINE NUMBER ONTO STACK
02E9 37              XPAH    P3
02EA AAFD            ILD     LSTK(P2)
02EC AAFD            ILD     LSTK(P2)
02EE 33              XPAL    P3
02EF C501            LD      @1(P1)
02F1 CBFF            ST      -1(P3)
02F3 C501            LD      @1(P1)
02F5 CBFE            ST      -2(P3)
02F7 C501            LD      @1(P1)          ;SKIP OVER LINE LENGTH
02F9 C401            LDI     1
02FB CAF5            ST      LISTNG(P2)      ;SET LISTING FLAG
02FD 90DE            JMP     X6              ;GO PRINT LINE NUMBER
02FF C400   LST2:    LDI     0
0301 CAF5            ST      LISTNG(P2)      ;CLEAR LISTING FLAG
0303 C402            JS      P3,NXT          ;GO TO NXT
030A 90D1   X6A:     JMP     X6
030C 90D1   E5:      JMP     E4
030E        LST3:    LDPI    P3,PUTC-1       ;POINT P3 AT PUTC
0314 06     LST4:    CSA
0315 D420            ANI     020
0317 98E6            JZ      LST2            ;IF TYPING, STOP
0319 C501            LD      @1(P1)          ;GET NEXT CHAR
031B E40D            XRI     0D              ;TEST FOR CR
031D 9805            JZ      LST5
031F E40D            XRI     0D              ;GET CHARACTER
0321 3F              XPPC    P3              ;PRINT CHARACTER
0322 90F0            JMP     LST4
0324 C40D   LST5:    LDI     0D              ;CARRIAGE RETURN
0326 3F              XPPC    P3
0327 C40A            LDI     0A              ;LINE FEED
0329 3F              XPPC    P3
032A 02              CCL
032B C447            LDI     L(LIST3)
032D CAFB            ST      PCLOW(P2)
032F C40C            LDI     H(LIST3)
0331 CAFA            ST      PCHIGH(P2)
0333 90AC            JMP     LST             ;GET NEXT LINE

            ;************************************
            ;*          ADD AND SUBTRACT        *
            ;************************************

0335 C410   ADD:     LDI     H(AESTK)        ;SET P3 TO CURRENT-
0337 37              XPAH    P3              ; STACK LOCATION
0338 BAFD            DLD     LSTK(P2)
033A BAFD            DLD     LSTK(P2)
033C 33              XPAL    P3
033D 02              CCL
033E C3FE            LD      -2(P3)          ;REPLACE TWO TOP ITEMS
0340 F300            ADD     0(P3)           ; ON STACK BY THEIR SUM
0342 CBFE            ST      -2(P3)
0344 C3FF            LD      -1(P3)
0346 F301            ADD     1(P3)
0348 CBFF            ST      -1(P3)
034A 90BE   X7:      JMP     X6A

034C C410   SUB:     LDI     H(AESTK)        ;SET P3 TO CURRENT
034E 37              XPAH    P3              ; STACK LOCATION
034F BAFD            DLD     LSTK(P2)
0351 BAFD            DLD     LSTK(P2)
0353 33              XPAL    P3
0354 03              SCL
0355 C3FE            LD      -2(P3)          ;REPLACE TWO TOP ITEMS
0357 FB00            CAD     0(P3)           ; ON STACK BY THEIR DIFFERENC
0359 CBFE            ST      -2(P3)
035B C3FF            LD      -1(P3)
035D FB01            CAD     1(P3)
035F CBFF            ST      -1(P3)
0361 90A7            JMP     X6A

            ;************************************
            ;*          NEGATE                  *
            ;************************************

0363 C410   NEG:     LDI     H(AESTK)        ;SET P3 TO CURRENT
0365 37              XPAH    P3              ; STACK LOCATION
0366 C2FD            LD      LSTK(P2)
0368 33              XPAL    P3
0369 03              SCL
036A C400            LDI     0
036C FBFE            CAD     -2(P3)          ;NEGATE TOP ITEM ON STACK
036E CBFE            ST      -2(P3)
0370 C400            LDI     0
0372 FBFF            CAD     -1(P3)
0374 CBFF            ST      -1(P3)
0376 90D2   X8:      JMP     X7
0378 9092   E6:      JMP     E5

            ;************************************
            ;*        MULTIPLY                  *
            ;************************************

                     .LOCAL
037A C410   MUL:     LDI     H(AESTK)        ;SET P3 TO CURRENT
037C 37              XPAH    P3              ; STACK LOCATION
037D C2FD            LD      LSTK(P2)
037F 33              XPAL    P3              ;DETERMINE SIGN OF PRODUCT,
0380 C3FF            LD      -1(P3)          ; SAVE IN TEMP(P2)
0382 E3FD            XOR     -3(P3)
0384 CAEA            ST      TEMP(P2)
0386 C3FF            LD      -1(P3)          ;CHECK FOR NEGATIVE
0388 940D            JP      $1              ; MULTIPLIER
038A 03              SCL
038B C400            LDI     0               ;IF NEGATIVE,
038D FBFE            CAD     -2(P3)          ; NEGATE
038F CBFE            ST      -2(P3)
0391 C400            LDI     0
0393 FBFF            CAD     -1(P3)
0395 CBFF            ST      -1(P3)
0397 C3FD   $1:      LD      -3(P3)          ;CHECK FOR NEGATIVE
0399 940D            JP      $2              ; MULTIPLICAND
039B 03              SCL
039C C400            LDI     0               ;IF NEGATIVE,
039E FBFC            CAD     -4(P3)          ; NEGATE
03A0 CBFC            ST      -4(P3)
03A2 C400            LDI     0
03A4 FBFD            CAD     -3(P3)
03A6 CBFD            ST      -3(P3)
03A8 C400   $2:      LDI     0               ;CLEAR WORKSPACE
03AA CB00            ST      0(P3)
03AC CB01            ST      1(P3)
03AE CB02            ST      2(P3)
03B0 CB03            ST      3(P3)
03B2 C410            LDI     16              ;SET COUNTER TO 16
03B4 CAEB            ST      NUM(P2)
03B6 C3FF   $LOOP:   LD      -1(P3)          ;ROTATE MULTIPLIER
03B8 1F              RRL                     ; RIGHT ONE BIT
03B9 CBFF            ST      -1(P3)
03BB C3FE            LD      -2(P3)
03BD 1F              RRL
03BE CBFE            ST      -2(P3)
03C0 06              CSA                     ;CHECK FOR CARRY BIT
03C1 9411            JP      $3              ;IF NOT SET, DON'T DO ADD
03C3 02              CCL
03C4 C302            LD      2(P3)           ;ADD MULTIPLICAND
03C6 F3FC            ADD     -4(P3)          ; INTO WORKSPACE
03C8 CB02            ST      2(P3)
03CA C303            LD      3(P3)
03CC F3FD            ADD     -3(P3)
03CE CB03            ST      3(P3)
03D0 9002            JMP     $3
03D2 90A4   E6A:     JMP     E6
03D4 02     $3:      CCL
03D5 C303            LD      3(P3)           ;SHIFT WORKSPACE RIGHT BY 1
03D7 1F              RRL
03D8 CB03            ST      3(P3)
03DA C302            LD      2(P3)
```

```
03DC 1F          RRL
03DD CB02        ST      2(P3)
03DF C301        LD      1(P3)
03E1 1F          RRL
03E2 CB01        ST      1(P3)
03E4 C300        LD      0(P3)
03E6 1F          RRL
03E7 CB00        ST      0(P3)
03E9 BAEB        DLD     NUM(P2)         ;DECREMENT COUNTER
03EB 9CC9        JNZ     $LOOP           ;LOOP IF NOT ZERO
03ED 9002        JMP     $4
03EF 9085  X9:   JMP     X8
03F1 C2EA  $4:   LD      TEMP(P2)        ;CHECK SIGN WORD
03F3 940D        JP      $EXIT           ;IF BIT7 = 1, NEGATE PRODUCT
03F5 03          SCL
03F6 C400        LDI     0
03F8 FB00        CAD     0(P3)
03FA CB00        ST      0(P3)
03FC C400        LDI     0
03FE FB01        CAD     1(P3)
0400 CB01        ST      1(P3)
0402 C300  $EXIT: LD     0(P3)           ;PUT PRODUCT ON TOP
0404 CBFC        ST      -4(P3)          ; OF STACK
0406 C301        LD      1(P3)
0408 CBFD        ST      -3(P3)
040A BAFD        DLD     LSTK(P2)        ;SUBTRACT 2 FROM
040C BAFD        DLD     LSTK(P2)        ; LSTK
040E 90DF        JMP     X9

                 ;*************************************
                 ;*              DIVIDE               *
                 ;*************************************

                 .LOCAL
0410 C410  DIV:  LDI     H(AESTK)
0412 37          XPAH    P3
0413 C2FD        LD      LSTK(P2)
0415 33          XPAL    P3
0416 C3FF        LD      -1(P3)          ;CHECK FOR DIVISION BY 0
0418 DBFE        OR      -2(P3)
041A 9C04        JNZ     $0
041C C40D        LDI     13
041E 90B2        JMP     E6A
0420 C3FD  $0:   LD      -3(P3)
0422 E3FF        XOR     -1(P3)
0424 CAEA        ST      TEMP(P2)        ;SAVE SIGN OF QUOTIENT
0426 C3FD        LD      -3(P3)          ;IS DIVIDEND POSITIVE?
0428 9411        JP      $POS            ;YES - JUMP
042A C400        LDI     0
042C 03          SCL
042D FBFC        CAD     -4(P3)          ;NO - NEGATE DIVIDEND,
042F CB03        ST      3(P3)           ; STORE IN RIGHT HALF
0431 C400        LDI     0               ; OF 32-BIT ACCUMULATOR
0433 FBFD        CAD     -3(P3)
0435 CB02        ST      2(P3)
0437 900A        JMP     $1
0439 90B4  X9A:  JMP     X9
043B C3FD  $POS: LD      -3(P3)          ;STORE NON-NEGATED DIVIDEND
043D CB02        ST      2(P3)           ; IN 32-BIT ACCUMULATOR
043F C3FC        LD      -4(P3)
0441 CB03        ST      3(P3)
0443 C3FF  $1:   LD      -1(P3)          ;CHECK FOR NEGATIVE DIVISOR
0445 940D        JP      $2
0447 C400        LDI     0               ;NEGATE DIVISOR
0449 03          SCL
044A FBFE        CAD     -2(P3)
044C CBFE        ST      -2(P3)
044E C400        LDI     0
0450 FBFF        CAD     -1(P3)
0452 CBFF        ST      -1(P3)
0454 C400  $2:   LDI     0               ;PUT ZERO IN:
0456 CB01        ST      1(P3)           ;  LEFT HALF OF 32-BIT ACC,
0458 CB00        ST      0(P3)
045A CAEB        ST      NUM(P2)         ;  THE COUNTER, AND
045C CBFD        ST      -3(P3)          ;  IN THE DIVIDEND, NOW USED
045E CBFC        ST      -4(P3)          ;  STORE THE QUOTIENT
0460 02    $LOOP: CCL                    ;BEGIN MAIN DIVIDE LOOP:
0461 C3FC        LD      -4(P3)          ;  SHIFT QUOTIENT LEFT,
0463 F3FC        ADD     -4(P3)
0465 CBFC        ST      -4(P3)
0467 C3FD        LD      -3(P3)
0469 F3FD        ADD     -3(P3)
046B CBFD        ST      -3(P3)
046D 02          CCL                     ;  SHIFT 32-BIT ACC LEFT,
046E C303        LD      3(P3)
0470 F303        ADD     3(P3)
0472 CB03        ST      3(P3)
0474 C302        LD      2(P3)
0476 F302        ADD     2(P3)
0478 CB02        ST      2(P3)
047A C301        LD      1(P3)
047C F301        ADD     1(P3)
047E CB01        ST      1(P3)
0480 C300        LD      (P3)
0482 F300        ADD     (P3)
0484 CB00        ST      (P3)
0486 03          SCL
0487 C301        LD      1(P3)           ;  SUBTRACT DIVISOR INTO
0489 FBFE        CAD     -2(P3)          ;   LEFT HALF OF ACC,
048B CB01        ST      1(P3)
048D C300        LD      (P3)
048F FBFF        CAD     -1(P3)
0491 CB00        ST      (P3)
0493 9411        JP      $ENT1           ;  IF RESULT IS NEGATIVE,
0495 02          CCL                     ;   RESTORE ORIGINAL CONTENTS
0496 C301        LD      1(P3)           ;   OF ACC BY ADDING DIVISOR
0498 F3FE        ADD     -2(P3)
049A CB01        ST      1(P3)
049C C300        LD      (P3)
049E F3FF        ADD     -1(P3)
04A0 CB00        ST      (P3)
04A2 9008        JMP     $3
04A4 9093  X9B:  JMP     X9A
04A6 C3FC  $ENT1: LD     -4(P3)          ;  ELSE IF RESULT POSITIVE,
04A8 DC01        ORI     1               ;   RECORD A 1 IN QUOTIENT
04AA CBFC        ST      -4(P3)          ;   W/O RESTORING THE ACC
04AC AAEB  $3:   ILD     NUM(P2)         ;  INCREMENT THE COUNTER
04AE E410        XRI     16              ;  ARE WE DONE?
04B0 9CAE        JNZ     $LOOP           ;  LOOP IF NOT DONE
04B2 C2EA        LD      TEMP(P2)        ;CHECK THE QUOTIENT'S SIGN,
04B4 940D        JP      $END            ;  NEGATING IF NECESSARY
04B6 C400        LDI     0
04B8 03          SCL
04B9 FBFC        CAD     -4(P3)
04BB CBFC        ST      -4(P3)
04BD C400        LDI     0
04BF FBFD        CAD     -3(P3)
04C1 CBFD        ST      -3(P3)
04C3 BAFD  $END: DLD     LSTK(P2)        ;DECREMENT THE STACK POINTER,
04C5 BAFD        DLD     LSTK(P2)
04C7 90DB        JMP     X9B             ;  AND EXIT

                 ;*************************************
                 ;*          STORE VARIABLE           *
                 ;*************************************

04C9 C410  STORE: LDI     H(AESTK)        ;SET P3 TO STACK
04CB 37          XPAH    P3
04CC C2FD        LD      LSTK(P2)
04CE 33          XPAL    P3
04CF C7FD        LD      @-3(P3)         ;GET VARIABLE INDEX
04D1 01          XAE                     ;PUT IN E REG
04D2 C301        LD      1(P3)
04D4 CA80        ST      EREG(P2)        ;STORE LOWER 8 BITS
04D6 02          CCL                     ;  INTO VARIABLE
04D7 40          LDE                     ;INCREMENT INDEX
04D8 F401        ADI     1
04DA 01          XAE
04DB C302        LD      2(P3)
04DD CA80        ST      EREG(P2)        ;STORE UPPER 8 BITS
04DF 33          XPAL    P3              ;  INTO VARIABLE
04E0 CAFD        ST      LSTK(P2)        ;UPDATE STACK POINTER
04E2 C400  X10:  JS      P3,EXECIL

                 ;*************************************
                 ;*     TEST FOR VARIABLE IN TEXT     *
                 ;*************************************

04E9 C501  TSTVAR: LD    @1(P1)
04EB E420        XRI     ' '             ;SLEW OFF SPACES
04ED 98FA        JZ      TSTVAR
04EF C1FF        LD      -1(P1)          ;GET CHARACTER IN QUESTION
04F1 03          SCL
04F2 FC5B        CAI     'Z'+1           ;SUBTRACT 'Z'+1
04F4 9405        JP      $FAIL           ;NOT VARIABLE IF POSITIVE
04F6 03          SCL
04F7 FCE6        CAI     'A'-'Z'-1       ;SUBTRACT 'A'
04F9 9412        JP      $MAYBE          ;IF POS, MAY BE VARIABLE
04FB C5FF  $FAIL: LD     @-1(P1)         ;BACKSPACE CURSOR
04FD C2FB        LD      PCLOW(P2)       ;GET TEST-FAIL ADDRESS
04FF 33          XPAL    P3              ;  FROM I.L. TABLE, PUT IT
0500 C2FA        LD      PCHIGH(P2)      ;  INTO I.L. PROGRAM COUNTER
0502 37          XPAH    P3
0503 C300        LD      (P3)
0505 CAFA        ST      PCHIGH(P2)
0507 C301        LD      1(P3)
0509 CAFB        ST      PCLOW(P2)
050B 90D5        JMP     X10
050D 01    $MAYBE: XAE                   ;SAVE VALUE (0-25)
050E C100        LD      (P1)            ;CHECK FOLLOWING CHAR
0510 03          SCL                     ;MUST NOT BE A LETTER
0511 FC5B        CAI     'Z'+1           ;  OTHERWISE WE'D BE LOOKING
0513 9405        JP      $OK             ;  AT A KEYWORD, NOT VARIABLE
0515 03          SCL
0516 FCE6        CAI     'A'-'Z'-1
0518 94E1        JP      $FAIL
051A C410  $OK:  LDI     H(AESTK)        ;SET P3 TO CURRENT
051C 37          XPAH    P3              ;  STACK LOCATION
051D AAFD        ILD     LSTK(P2)        ;INCR STACK POINTER
051F 33          XPAL    P3
0520 02          CCL                     ;DOUBLE VARIABLE INDEX
0521 40          LDE
0522 70          ADE
0523 CBFF        ST      -1(P3)          ;PUT INDEX ON STACK
0525 C402        LDI     2               ;INCREMENT I.L. PC, SKIPPING
0527 02          CCL                     ;  OVER TEST-FAIL ADDRESS
0528 F2FB        ADD     PCLOW(P2)
052A CAFB        ST      PCLOW(P2)
052C C400        LDI     0
052E F2FA        ADD     PCHIGH(P2)
0530 CAFA        ST      PCHIGH(P2)
0532 90AE        JMP     X10

                 ;*************************************
                 ;*    IND -- EVALUATE A VARIABLE     *
                 ;*************************************

0534 C410  IND:  LDI     H(AESTK)        ;SET P3 TO STACK
0536 37          XPAH    P3
0537 AAFD        ILD     LSTK(P2)
0539 33          XPAL    P3
053A C3FE        LD      -2(P3)          ;GET INDEX OFF TOP
053C 01          XAE                     ;PUT INDEX IN E REG
053D C280        LD      EREG(P2)        ;GET LOWER 8 BITS
053F CBFE        ST      -2(P3)          ;SAVE ON STACK
0541 02          CCL
0542 40          LDE                     ;INCREMENT E REG
0543 F401        ADI     1
0545 01          XAE
0546 C280        LD      EREG(P2)        ;GET UPPER 8 BITS
0548 CBFF        ST      -1(P3)          ;SAVE ON STACK
054A 9096  X11:  JMP     X10

                 ;*************************************
                 ;*    RELATIONAL OPERATORS           *
                 ;*************************************

054C C401  EQ:   LDI     1               ;EACH RELATIONAL OPERATOR
054E 9012        JMP     CMP             ;  LOADS A NUMBER USED LATER
0550 C402  NEQ:  LDI     2               ;  AS A CASE SELECTOR, AFTER
0552 900E        JMP     CMP             ;  THE TWO OPERANDS ARE COM-
0554 C403  LSS:  LDI     3               ;  PARED.  BASED ON THE COM-
0556 900A        JMP     CMP             ;  PARISON, FLAGS ARE SET THAT
0558 C404  LEQ:  LDI     4               ;  ARE EQUIVALENT TO THOSE SET
055A 9006        JMP     CMP             ;  BY THE 'CMP' INSTRUCTION IN
```

```
055C C405   GTR:    LDI     5               ; THE PDP-11.  THESE PSEUDO-
055E 9002           JMP     CMP             ; FLAGS ARE USED TO DETERMINE
0560 C406   GEQ:    LDI     6               ; WHETHER THE PARTICULAR
                                            ; RELATION IS SATISFIED OR NO
0562 CAEB   CMP:    ST      NUM(P2)
0564 C410           LDI     H(AESTK)        ;SET P3 -> ARITH STACK
0566 37             XPAH    P3
0567 BAFD           DLD     LSTK(P2)
0569 BAFD           DLD     LSTK(P2)
056B 33             XPAL    P3
056C 03             SCL
056D C3FE           LD      -2(P3)          ;SUBTRACT THE TWO OPERANDS,
056F FB00           CAD     (P3)            ; STORING RESULT IN LO & HI
0571 CAEF           ST      LO(P2)
0573 C3FF           LD      -1(P3)
0575 FB01           CAD     1(P3)
0577 CAEE           ST      HI(P2)
0579 E3FF           XOR     -1(P3)          ;OVERFLOW OCCURS IF SIGNS OF
057B 01             XAE                     ; RESULT AND 1ST OPERAND
057C C3FF           LD      -1(P3)          ; DIFFER, AND SIGNS OF THE
057E E301           XOR     1(P3)           ; TWO OPERANDS DIFFER
0580 50             ANE                     ;BIT 7 EQUIVALENT TO V FLAG
0581 E2EE           XOR     HI(P2)          ;BIT 7 EQUIVALENT TO N XOR V
0583 CAEA           ST      TEMP(P2)        ;STORE IN TEMP
0585 C2EE           LD      HI(P2)          ;DETERMINE IF RESULT WAS ZERO
0587 DAEF           OR      LO(P2)
0589 9802           JZ      SETZ            ;IF RESULT=0, SET Z FLAG
058B C480           LDI     080             ; ELSE CLEAR Z FLAG
058D E480   SETZ:   XRI     080
058F 01             XAE                     ;BIT 7 OF EX = Z FLAG

0590 BAEB           DLD     NUM(P2)         ;TEST FOR =
0592 9C05           JNZ     NEQ1
0594 40             LDE                     ; EQUAL IF Z = 1
0595 902B           JMP     CMP1
0597 90B1   X12:    JMP     X11
0599 BAEB   NEQ1:   DLD     NUM(P2)         ;TEST FOR <>
059B 9C05           JNZ     LSS1
059D 40             LDE                     ; NOT EQUAL IF Z = 0
059E E480           XRI     080
05A0 9020           JMP     CMP1
05A2 BAEB   LSS1:   DLD     NUM(P2)         ;TEST FOR <
05A4 9C04           JNZ     LEQ1
05A6 C2EA           LD      TEMP(P2)        ; LESS THAN IF (N XOR V)=1
05A8 9018           JMP     CMP1
05AA BAEB   LEQ1:   DLD     NUM(P2)         ;TEST FOR <=
05AC 9C05           JNZ     GTR1
05AE 40             LDE                     ; LESS THAN OR EQUAL
05AF DAEA           OR      TEMP(P2)        ;  IF (Z OR (N XOR V))=1
05B1 900F           JMP     CMP1
05B3 BAEB   GTR1:   DLD     NUM(P2)         ;TEST FOR >
05B5 9C07           JNZ     GEQ1
05B7 40             LDE                     ; GREATER THAN
05B8 DAEA           OR      TEMP(P2)        ;  IF (Z OR (N XOR V))=0
05BA E480           XRI     080
05BC 9004           JMP     CMP1
05BE C2EA   GEQ1:   LD      TEMP(P2)        ;GREATER THAN OR EQUAL
05C0 E480           XRI     080             ; IF (N XOR V)=0
05C2 9404   CMP1:   JP      FALSE           ;IS RELATION SATISFIED?
05C4 C401           LDI     1               ;YES - PUSH 1 ON STACK
05C6 9002           JMP     CMP2
05C8 C400   FALSE:  LDI     0               ;NO - PUSH 0 ON STACK
05CA CBFE   CMP2:   ST      -2(P3)
05CC C400           LDI     0
05CE CBFF           ST      -1(P3)
05D0 C400           JS      P3,RTN          ;DO AN I.L.  RETURN
05D2 90BE           JMP     X12

                    ;************************************
                    ;*   IF STATEMENT TEST FOR ZERO     *
                    ;************************************

05D9 C2EF   CMPR:   LD      LO(P2)          ;GET LOW & HI BYTES OF EXPR.
05DB DAEE           OR      HI(P2)          ;TEST IF EXPRESSION IS ZERO
05DD 9802           JZ      FAIL            ;YES - IT IS
05DF 90B6           JMP     X12             ;NO - IT ISN'T SO CONTINUE
05E1 C501   FAIL:   LD      @1(P1)          ;SKIP TO NEXT LINE IN PROGRAM
05E3 E40D           XRI     0D              ; (I.E.  TIL NEXT CR)
05E5 9CFA           JNZ     FAIL
05E7 C402           JS      P3,NXT          ;CALL NXT AND RETURN
05EE 90A7   X12A:   JMP     X12

                    ;************************************
                    ;*        AND, OR, & NOT            *
                    ;************************************

                    .LOCAL
05F0 C401   ANDOP:  LDI     1               ;EACH OPERATION HAS ITS
05F2 9006           JMP     $1              ; OWN CASE SELECTOR.
05F4 C402   OROP:   LDI     2
05F6 9002           JMP     $1
05F8 C403   NOTOP:  LDI     3
05FA CAEB   $1:     ST      NUM(P2)
05FC C410           LDI     H(AESTK)        ;SET P3 -> ARITH.  STACK
05FE 37             XPAH    P3
05FF BAFD           DLD     LSTK(P2)
0601 BAFD           DLD     LSTK(P2)
0603 33             XPAL    P3
0604 BAEB           DLD     NUM(P2)         ;TEST FOR 'AND'
0606 9C0E           JNZ     $OR
0608 C301           LD      1(P3)           ;REPLACE TWO TOP ITEMS ON
060A D3FF           AND     -1(P3)          ; STACK BY THEIR 'AND'
060C CBFF           ST      -1(P3)
060E C300           LD      0(P3)
0610 D3FE           AND     -2(P3)
0612 CBFE           ST      -2(P3)
0614 90D8           JMP     X12A
0616 BAEB   $OR:    DLD     NUM(P2)         ;TEST FOR 'OR'
0618 9C0E           JNZ     $NOT
061A C301           LD      1(P3)           ;REPLACE TWO TOP ITEMS ON
061C DBFF           OR      -1(P3)          ; STACK BY THEIR 'OR'
061E CBFF           ST      -1(P3)
0620 C300           LD      0(P3)
0622 DBFE           OR      -2(P3)
0624 CBFE           ST      -2(P3)
0626 90C6           JMP     X12A
0628 C701   $NOT:   LD      @1(P3)          ;'NOT' OPERATION
062A E4FF           XRI     0FF
062C CBFF           ST      -1(P3)          ;REPLACE TOP ITEM ON STACK
062E C701           LD      @1(P3)          ; BY ITS ONE'S COMPLEMENT
0630 E4FF           XRI     0FF
0632 CBFF           ST      -1(P3)
0634 33             XPAL    P3
0635 CAFD           ST      LSTK(P2)        ;STACK POINTER FIXUP
0637 90B5   X12B:   JMP     X12A

                    ;************************************
                    ;*     EXCHANGE CURSOR WITH RAM     *
                    ;************************************

0639 C2F1   XCHGP1: LD      P1LOW(P2)       ;THIS ROUTINE IS HANDY WHEN
063B 31             XPAL    P1              ; EXECUTING AN 'INPUT' STMT
063C CAF1           ST      P1LOW(P2)       ; IT EXCHANGES THE CURRENT
063E C2F0           LD      P1HIGH(P2)      ; TEXT CURSOR WITH ONE SAVED
0640 35             XPAH    P1              ; IN RAM
0641 CAF0           ST      P1HIGH(P2)
0643 3F             XPPC    P3

                    ;************************************
                    ;*          CHECK RUN MODE          *
                    ;************************************

0644 C2F4   CKMODE: LD      RUNMOD(P2)      ;THIS ROUTINE CAUSES AN ERROR
0646 9801           JZ      CK1             ; IF CURRENTLY IN EDIT MODE
0648 3F             XPPC    P3
0649 C403   CK1:    LDI     3
064B CAEB   E8:     ST      NUM(P2)         ;ERROR IF RUN MODE = 0
064D C402           JS      P3,ERR2         ;MINOR KLUGE

                    ;************************************
                    ;*      GET HEXADECIMAL NUMBER      *
                    ;************************************

                    .LOCAL
0654 AAFD   HEX:    ILD     LSTK(P2)        ;POINT P3 AT ARITH STACK
0656 AAFD           ILD     LSTK(P2)
0658 33             XPAL    P3
0659 C410           LDI     H(AESTK)
065B 37             XPAH    P3
065C C400           LDI     0               ;NUMBER INITIALLY ZERO
065E CBFF           ST      -1(P3)          ;PUT IT ON STACK
0660 CBFE           ST      -2(P3)
0662 CAEB           ST      NUM(P2)         ;ZERO NUMBER OF DIGITS
0664 C501   $SKIP:  LD      @1(P1)          ;SKIP ANY SPACES
0666 E420           XRI     ' '
0668 98FA           JZ      $SKIP
066A C5FF           LD      @-1(P1)
066C C100   $LOOP:  LD      (P1)            ;GET A CHARACTER
066E 03             SCL
066F FC3A           CAI     '9'+1           ;CHECK FOR A NUMERIC CHAR
0671 9409           JP      $LETR
0673 03             SCL
0674 FCF6           CAI     '0'-'9'-1       ;IF NUMERIC, SHIFT NUMBER
0676 9413           JP      $ENTER          ; AND ADD NEW HEX DIGIT
0678 9032           JMP     $END
067A 90BB   X12C:   JMP     X12B
067C 03     $LETR:  SCL                     ;CHECK FOR HEX LETTER
067D FC0D           CAI     'G'-'9'-1
067F 942B           JP      $END
0681 03             SCL
0682 FCFA           CAI     'A'-'G'
0684 9402           JP      $OK
0686 9024           JMP     $END
0688 02     $OK:    CCL                     ;ADD 10 TO GET TRUE VALUE
0689 F40A           ADI     10              ; OF LETTER
068B 01     $ENTER: XAE                     ;NEW DIGIT IN EX REG
068C C404           LDI     4               ;SET SHIFT COUNTER
068E CAEA           ST      TEMP(P2)
0690 CAEB           ST      NUM(P2)         ;DIGIT COUNT IS NON-ZERO
0692 C3FE   $SHIFT: LD      -2(P3)          ;SHIFT NUMBER LEFT BY 4
0694 02             CCL
0695 F3FE           ADD     -2(P3)
0697 CBFE           ST      -2(P3)
0699 C3FF           LD      -1(P3)
069B F3FF           ADD     -1(P3)
069D CBFF           ST      -1(P3)
069F BAEA           DLD     TEMP(P2)
06A1 9CEF           JNZ     $SHIFT
06A3 C3FE           LD      -2(P3)          ;ADD NEW DIGIT
06A5 58             ORE                     ; INTO NUMBER
06A6 CBFE           ST      -2(P3)
06A8 C501           LD      @1(P1)          ;ADVANCE THE CURSOR
06AA 90C0           JMP     $LOOP           ;GET NEXT CHAR
06AC C2EB   $END:   LD      NUM(P2)         ;CHECK IF THERE WERE
06AE 9C87           JNZ     X12B            ; MORE THAN 0 CHARACTERS
06B0 C405           LDI     5               ;ERROR IF THERE WERE NONE
06B2 9097   E8B:    JMP     E8

                    ;************************************
                    ;*     TEST FOR NUMBER IN TEXT      *
                    ;************************************

                    ; THIS ROUTINE TESTS FOR A NUMBER IN THE TEXT.  IF NO
                    ; NUMBER IS FOUND, I.L. CONTROL PASSES TO THE ADDRESS
                    ; INDICATED IN THE 'TSTN' INSTRUCTION.  OTHERWISE, THE
                    ; NUMBER IS SCANNED AND PUT ON THE ARITHMETIC STACK,
                    ; WITH I.L. CONTROL PASSING TO THE NEXT INSTRUCTION.

                    .LOCAL
06B4 C501   TSTNUM: LD      @1(P1)
06B6 E420           XRI     ' '             ;SKIP OVER ANY SPACES
06B8 98FA           JZ      TSTNUM
06BA C5FF           LD      @-1(P1)         ;GET FIRST CHAR
06BC 03             SCL                     ;TEST FOR DIGIT
06BD FC3A           CAI     '9'+1
06BF 9405           JP      $ABORT
06C1 03             SCL
06C2 FCF6           CAI     '0'-'9'-1
06C4 9421           JP      $1
06C6 C2FB   $ABORT: LD      PCLOW(P2)       ;GET TEST-FAIL ADDRESS
06C8 33             XPAL    P3              ; FROM I.L.  TABLE
06C9 C2FA           LD      PCHIGH(P2)
06CB 37             XPAH    P3
```

```
06CC C300          LD      (P3)            ;PUT TEST-FAIL ADDRESS
06CE CAFA          ST      PCHIGH(P2)      ; INTO I.L. PC
06D0 C301          LD      1(P3)
06D2 CAFB          ST      PCLOW(P2)
06D4 90A4          JMP     X12C
06D6 C402  $RET:   LDI     2               ;SKIP OVER ONE IL INSTRUCTION
06D8 02            CCL                     ; IF NUMBER IS DONE
06D9 F2FB          ADD     PCLOW(P2)
06DB CAFB          ST      PCLOW(P2)
06DD C400          LDI     0
06DF F2FA          ADD     PCHIGH(P2)
06E1 CAFA          ST      PCHIGH(P2)
06E3 9095  X13:    JMP     X12C
06E5 90CB  E8A:    JMP     E8B
06E7 01    $1:     XAE                     ;SAVE DIGIT IN EX REG
06E8 C410          LDI     H(AESTK)        ;POINT P3 AT AE STACK
06EA 37            XPAH    P3
06EB AAFD          ILD     LSTK(P2)
06ED AAFD          ILD     LSTK(P2)
06EF 33            XPAL    P3
06F0 C400          LDI     0
06F2 CBFF          ST      -1(P3)
06F4 40            LDE
06F5 CBFE          ST      -2(P3)
06F7 C501  $LOOP:  LD      @1(P1)          ;GET NEXT CHAR
06F9 C100          LD      (P1)
06FB 03            SCL                     ;TEST IF IT IS DIGIT
06FC FC3A          CAI     '9'+1
06FE 94D6          JP      $RET            ;RETURN IF IT ISN'T
0700 03            SCL
0701 FCF6          CAI     '0'-'9'-1
0703 9402          JP      $2
0705 90CF          JMP     $RET
0707 01    $2:     XAE                     ;SAVE DIGIT
0708 C3FF          LD      -1(P3)          ;PUT RESULT IN SCRATCH SPACE
070A CB01          ST      1(P3)
070C C3FE          LD      -2(P3)
070E CB00          ST      (P3)
0710 C402          LDI     2
0712 CAEA          ST      TEMP(P2)        ;MULTIPLY RESULT BY 10
0714 02    $SHIFT: CCL                     ;FIRST MULTIPLY BY 4
0715 C3FE          LD      -2(P3)
0717 F3FE          ADD     -2(P3)
0719 CBFE          ST      -2(P3)
071B C3FF          LD      -1(P3)
071D F3FF          ADD     -1(P3)
071F CBFF          ST      -1(P3)
0721 BAEA          DLD     TEMP(P2)
0723 9CEF          JNZ     $SHIFT
0725 02            CCL                     ;THEN ADD OLD RESULT,
0726 C3FE          LD      -2(P3)          ; SO WE HAVE RESULT * 5
0728 F300          ADD     (P3)
072A CBFE          ST      -2(P3)
072C C3FF          LD      -1(P3)
072E F301          ADD     1(P3)
0730 CBFF          ST      -1(P3)
0732 02            CCL                     ;THEN MULTIPLY BY TWO
0733 C3FE          LD      -2(P3)
0735 F3FE          ADD     -2(P3)
0737 CBFE          ST      -2(P3)
0739 C3FF          LD      -1(P3)
073B F3FF          ADD     -1(P3)
073D CBFF          ST      -1(P3)
073F 02            CCL                     ;THEN ADD IN NEW DIGIT
0740 40            LDE
0741 F3FE          ADD     -2(P3)
0743 CBFE          ST      -2(P3)
0745 C400          LDI     0
0747 F3FF          ADD     -1(P3)
0749 CBFF          ST      -1(P3)
074B 94AA          JP      $LOOP           ;REPEAT IF NO OVERFLOW
074D C406          LDI     6
074F 9094  E9:     JMP     E8A             ;ELSE REPORT ERROR
0751 9090  X14:    JMP     X13

                ;*************************************
                ;*    GET LINE FROM TELETYPE         *
                ;*************************************

                .LOCAL
0753       GETL:   LDPI    P1,LBUF         ;SET P1 TO LBUF
0759 C400          LDI     0               ;CLEAR NO. OF CHAR
075B CAE7          ST      CHRNUM(P2)
075D               LDPI    P3,PUTC-1       ;POINT P3 AT PUTC ROUTINE
0763 C2F4          LD      RUNMOD(P2)      ;PRINT '? ' IF RUNNING
0765 9808          JZ      $0              ; (I.E. DURING 'INPUT')
0767 C43F          LDI     '?'
0769 3F            XPPC    P3
076A C420          LDI     ' '
076C 3F            XPPC    P3
076D 9003          JMP     $1
076F C43E  $0:     LDI     '>'             ;OTHERWISE PRINT '>'
0771 3F            XPPC    P3
0772 C40F  $1:     JS      P3,GECO         ;GET CHARACTER
0779 C4BD          LDI     L(PUTC)-1       ;POINT P3 AT PUTC AGAIN
077B 33            XPAL    P3
077C 40            LDE                     ;GET TYPED CHAR
077D 98F3          JZ      $1              ;IGNORE NULLS
077F E40A          XRI     0A              ;IGNORE LINE FEED
0781 98EF          JZ      $1
0783 40            LDE
0784 E40D          XRI     0D              ;CHECK FOR CR
0786 9850          JZ      $CR
0788 40            LDE
0789 E45F          XRI     'O'+010         ;CHECK FOR SHIFT/O
078B 9841          JZ      $RUB
078D 40            LDE                     ;CHECK FOR CTRL/H
078E E408          XRI     8
0790 9836          JZ      $XH
0792 40            LDE
0793 E415          XRI     015             ;CHECK FOR CTRL/U
0795 980F          JZ      $XU
0797 40            LDE
0798 E403          XRI     3               ;CHECK FOR CTRL/C
079A 9C1A          JNZ     $ENTER
079C C45E          LDI     "^"             ;ECHO CONTROL/C AS ^C
079E 3F            XPPC    P3
079F C443          LDI     'C'
07A1 3F            XPPC    P3
07A2 C40E          LDI     14              ;CAUSE A BREAK
07A4 90A9          JMP     E9
07A6 C45E  $XU:    LDI     "^"             ;ECHO CONTROL/U AS ^U
07A8 3F            XPPC    P3
07A9 C455          LDI     'U'
07AB 3F            XPPC    P3
07AC C40D          LDI     0D              ;PRINT CR/LF
07AE 3F            XPPC    P3
07AF C40A          LDI     0A
07B1 3F            XPPC    P3
07B2 909F  $2:     JMP     GETL            ;GO GET ANOTHER LINE
07B4 909B  X15:    JMP     X14
07B6 40    $ENTER: LDE
07B7 CD01          ST      @1(P1)          ;PUT CHAR IN LBUF
07B9 AAE7          ILD     CHRNUM(P2)      ;INCREMENT CHRNUM
07BB E448          XRI     72              ;IF=72, LINE FULL
07BD 9CB3          JNZ     $1
07BF C40D          LDI     0D
07C1 01            XAE                     ;SAVE CARRIAGE RET
07C2 40            LDE
07C3 3F            XPPC    P3              ;PRINT IT
07C4 9012          JMP     $CR             ;STORE IT IN LBUF
07C6 9087  E10:    JMP     E9
07C8 C420  $XH:    LDI     ' '             ;BLANK OUT THE CHARACTER
07CA 3F            XPPC    P3
07CB C408          LDI     8               ;PRINT ANOTHER BACKSPACE
07CD 3F            XPPC    P3
07CE C2E7  $RUB:   LD      CHRNUM(P2)
07D0 98A0          JZ      $1
07D2 BAE7          DLD     CHRNUM(P2)      ;ONE LESS CHAR
07D4 C5FF          LD      @-1(P1)         ;BACKSPACE CURSOR
07D6 909A          JMP     $1
07D8 40    $CR:    LDE
07D9 CD01          ST      @1(P1)          ;STORE CR IN LBUF
07DB C40A          LDI     0A              ;PRINT LINE FEED
07DD 3F            XPPC    P3
07DE C410          LDI     H(LBUF)         ;SET P1 TO BEGIN-
07E0 35            XPAH    P1              ; NING OF LBUF
07E1 C4D6          LDI     L(LBUF)
07E3 31            XPAL    P1
07E4 90CE  X16:    JMP     X15

                ;*************************************
                ;*     EVAL -- GET MEMORY CONTENTS   *
                ;*************************************

                ; THIS ROUTINE IMPLEMENTS THE '@' OPERATOR IN EXPRESSIONS

07E6 C410  EVAL:   LDI     H(AESTK)
07E8 37            XPAH    P3
07E9 C2FD          LD      LSTK(P2)
07EB 33            XPAL    P3              ;P3 -> ARITH STACK
07EC C3FF          LD      -1(P3)          ;GET ADDR OFF STACK,
07EE 35            XPAH    P1              ; AND INTO P1,
07EF 01            XAE                     ; SAVING OLD P1 IN EX & LO
07F0 C3FE          LD      -2(P3)
07F2 31            XPAL    P1
07F3 CAEF          ST      LO(P2)
07F5 C100          LD      0(P1)           ;GET MEMORY CONTENTS,
07F7 CBFE          ST      -2(P3)          ; SHOVE ONTO STACK
07F9 C400          LDI     0
07FB CBFF          ST      -1(P3)          ;HIGH ORDER 8 BITS ZEROED
07FD C2EF          LD      LO(P2)
07FF 31            XPAL    P1              ;RESTORE ORIGINAL P1
0800 40            LDE
0801 35            XPAH    P1
0802 90B0          JMP     X15

                ;*************************************
                ;*    MOVE -- STORE INTO MEMORY      *
                ;*************************************

                ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
                ;    '@' FACTOR '=' REL-EXP

0804 C410  MOVE:   LDI     H(AESTK)
0806 37            XPAH    P3
0807 C2FD          LD      LSTK(P2)
0809 33            XPAL    P3              ;P3 -> ARITH STACK
080A C7FE          LD      @-2(P3)         ;GET BYTE TO BE MOVED
080C 01            XAE
080D C7FF          LD      @-1(P3)         ;NOW GET ADDRESS INTO P3
080F CAEA          ST      TEMP(P2)
0811 C7FF          LD      @-1(P3)
0813 33            XPAL    P3
0814 CAFD          ST      LSTK(P2)        ;STACK PTR UPDATED NOW
0816 C2EA          LD      TEMP(P2)
0818 37            XPAH    P3
0819 40            LDE
081A CB00          ST      0(P3)           ;MOVE THE BYTE INTO MEMORY
081C 90C6  X17:    JMP     X16
081E 90A6  E11:    JMP     E10

                ;*************************************
                ;*          TEXT EDITOR              *
                ;*************************************

                ;INPUTS TO THIS ROUTINE: POINTER TO LINE BUFFER IN P1LOW &
                ; P1HIGH.  P1 POINTS TO THE INSERTION POINT IN THE TEXT.
                ; THE A.E. STACK HAS THE LINE NUMBER ON IT (STACK POINTER
                ; IS ALREADY POPPED).

                ;EACH LINE IN THE NIBL TEXT IS STORED IN THE FOLLOWING
                ; FORMAT: TWO BYTES CONTAINING THE LINE NUMBER (IN BINARY,
                ; HIGH ORDER BYTE FIRST), THEN ONE BYTE CONTAINING THE
                ; LENGTH OF THE LINE, AND FINALLY THE LINE ITSELF FOLLOWED
                ; BY A CARRIAGE RETURN.  THE LAST LINE IN THE TEXT IS
                ; FOLLOWED BY TWO CONSECUTIVE BYTES OF X'FF.

                .LOCAL
0820 C410  INSRT:  LDI     H(AESTK)        ;POINT P3 AT AE STACK,
0822 37            XPAH    P3              ; WHICH HAS THE LINE #
0823 C2FD          LD      LSTK(P2)        ; ON IT
0825 33            XPAL    P3
0826 C301          LD      1(P3)           ;SAVE NEW LINE'S NUMBER
0828 CAF7          ST      HILINE(P2)
082A C300          LD      0(P3)
```

```
082C CAF8          ST    LOLINE(P2)
082E C2F1          LD    P1LOW(P2)      ;PUT POINTER TO LBUF INTO P3
0830 33            XPAL  P3
0831 C2F0          LD    P1HIGH(P2)
0833 37            XPAH  P3
0834 C404          LDI   4              ;INITIALLY LENGTH OF NEW LINE
0836 CAE7          ST    CHRNUM(P2)     ; = 4.  ADD 1 TO LENGTH FOR
0838 C701   $1:    LD    @1(P3)         ; EACH CHAR IN LINE UP TO, BU
083A E40D          XRI   0D             ; NOT INCLUDING, CARR. RETURN
083C 9804          JZ    $2
083E AAE7          ILD   CHRNUM(P2)
0840 90F6          JMP   $1
0842 C2E7   $2:    LD    CHRNUM(P2)     ;IF LENGTH STILL 4, WE'LL DEL
0844 E404          XRI   4              ; A LINE, SO SET LENGTH = 0
0846 9C02          JNZ   $3
0848 CAE7          ST    CHRNUM(P2)
084A C2E7   $3:    LD    CHRNUM(P2)     ;PUT NEW LINE LENGTH IN EX
084C 01            XAE
084D C2F2          LD    LABLHI(P2)     ;IS NEW LINE REPLACING OLD?
084F 9406          JP    $4             ;YES - DO REPLACE
0851 D47F          ANI   07F            ;NO - WE'LL INSERT LINE HERE,
0853 CAF2          ST    LABLHI(P2)     ; WHERE FNDLBL GOT US.
0855 9018          JMP   $MOVE          ;BUT FIRST MAKE ROOM
0857 C503   $4:    LD    @3(P1)         ;SKIP LINE # AND LENGTH
0859 40            LDE                  ;EX, NOW HOLDING NEW LINE
085A 02            CCL                  ; LENGTH, WILL SOON HOLD
085B F4FC          ADI   -4             ; DISPLACEMENT OF LINES
085D 01            XAE                  ; TO BE MOVED
085E C501   $5:    LD    @1(P1)         ;SUBTRACT 1 FROM DISPLACEMENT
0860 E40D          XRI   0D             ; FOR EACH CHAR IN LINE BEING
0862 9808          JZ    $MOVE          ; REPLACED
0864 40            LDE
0865 02            CCL
0866 F4FF          ADI   -1
0868 01            XAE
0869 90F3          JMP   $5
086B 90AF   X19:   JMP   X17
086D 90AF   E12:   JMP   E11
086F 40     $MOVE: LDE                  ;IF DISPLACEMENT AND LENGTH
0870 DAE7          OR    CHRNUM(P2)     ; OF NEW LINE ARE 0, RETURN
0872 98F7          JZ    X19
0874 C47A          LDI   L(DOSTAK)      ;CLEAR SOME STACKS
0876 CAFF          ST    DOPTR(P2)
0878 C46A          LDI   L(SBRSTK)
087A CAFC          ST    SBRPTR(P2)
087C C48A          LDI   L(FORSTK)
087E CAFE          ST    FORPTR(P2)
0880 40            LDE
0881 9860          JZ    $ADD           ;DON'T NEED TO MOVE LINES
0883 9410          JP    $UP            ;SKIP IF DISPLACEMENT POSITIV
0885 C100   $DOWN: LD    0(P1)          ;NEGATIVE DISPLACEMENT:
0887 C980          ST    EREG(P1)       ;DO;
0889 C501          LD    @1(P1)         ;    M(P1+DISP) = M(P1);
088B 94F8          JP    $DOWN          ;    P1 = P1+1;
088D C100          LD    0(P1)          ;UNTIL M(P1)<0 & M(P1-1)<0;
088F 94F4          JP    $DOWN
0891 C980          ST    EREG(P1)       ;M(P1+DISP) = M(P1);
0893 904E          JMP   $ADD
0895 C1FE   $UP:   LD    -2(P1)         ;POSITIVE DISPLACEMENT:
0897 CAEA          ST    TEMP(P2)       ;FLAG BEGINNING OF MOVE WITH
0899 C4FF          LDI   -1             ; A -1 FOLLOWED BY 80, WHICH
089B C9FE          ST    -2(P1)         ; CAN NEVER APPEAR IN A
089D C450          LDI   80             ; NIBL TEXT
089F C9FF          ST    -1(P1)
08A1 C501   $UP1:  LD    @1(P1)         ;ADVANCE P1 TO END OF TEXT
08A3 94FC          JP    $UP1
08A5 C100          LD    0(P1)
08A7 94F8          JP    $UP1
08A9 35            XPAH  P1             ;SAVE P1 IN LO, HI
08AA CAEE          ST    HI(P2)
08AC 35            XPAH  P1
08AD 31            XPAL  P1
08AE CAEF          ST    LO(P2)
08B0 31            XPAL  P1
08B1 C2EF          LD    LO(P2)         ;ADD DISPLACEMENT TO
08B3 02            CCL                  ; VALUE OF P1, TO CHECK
08B4 70            ADE                  ; WHETHER WE'RE OUT OF
08B5 C400          LDI   0              ; RAM FOR USER'S PROGRAM
08B7 F2EE          ADD   HI(P2)
08B9 E2EE          XOR   HI(P2)
08BB D4F0          ANI   0F0
08BD 9803          JZ    $UP2
08BF C400          LDI   0              ;IF OUT OF RAM, CHANGE
08C1 01            XAE                  ; DISPLACEMENT TO ZERO
08C2 C4FF   $UP2:  LDI   -1
08C4 C980   $UP3:  ST    EREG(P1)       ;MOVE TEXT UP UNTIL WE REACH
08C6 C5FF          LD    @-1(P1)        ; THE FLAGS SET ABOVE
08C8 94FA          JP    $UP3
08CA C101          LD    1(P1)
08CC E450          XRI   80
08CE 9804          JZ    $UP4
08D0 C100          LD    0(P1)
08D2 90F0          JMP   $UP3
08D4 C2EA   $UP4:  LD    TEMP(P2)       ;RESTORE THE FLAGGED LOCATION
08D6 C900          ST    0(P1)          ; TO THEIR ORIGINAL VALUES
08D8 C40D          LDI   0D
08DA C901          ST    1(P1)
08DC 40            LDE                  ;IF DISPLACEMENT = 0, WE'RE
08DD 9C04          JNZ   $ADD           ; OUT OF RAM, SO REPORT ERROR
08DF C402          LDI   2
08E1 908A   E12A:  JMP   E12
08E3 C2E7   $ADD:  LD    CHRNUM(P2)     ;INSERT NEW LINE
08E5 9884   X19A:  JZ    X19            ; UNLESS LENGTH IS ZERO
08E7 C2F1          LD    P1LOW(P2)      ;POINT P1 AT LINE BUFFER
08E9 31            XPAL  P1
08EA C2F0          LD    P1HIGH(P2)
08EC 35            XPAH  P1
08ED C2F3          LD    LABLLO(P2)     ;POINT P3 AT INSERTION PLACE
08EF 33            XPAL  P3
08F0 C2F2          LD    LABLHI(P2)
08F2 37            XPAH  P3
08F3 C2F7          LD    HILINE(P2)     ;PUT LINE NUMBER INTO TEXT
08F5 CF01          ST    @1(P3)
08F7 C2F8          LD    LOLINE(P2)
08F9 CF01          ST    @1(P3)
08FB C2E7          LD    CHRNUM(P2)     ;STORE LINE LENGTH IN TEXT
08FD CF01          ST    @1(P3)
08FF C501   $ADD1: LD    @1(P1)         ;PUT REST OF CHARS
0901 CF01          ST    @1(P3)         ; (INCLUDING CR) INTO TEXT
0903 E40D          XRI   0D
0905 9CF8          JNZ   $ADD1
0907 90DC          JMP   X19A           ;RETURN
0909 C400   X20:   JS    P3,EXECIL
0910 90CF   E13:   JMP   E12A

                   ;************************************
                   ;*       POP ARITHMETIC STACK       *
                   ;************************************

0912 BAFD   POPAE: DLD   LSTK(P2)       ;THIS ROUTINE POP THE A.E.
0914 BAFD          DLD   LSTK(P2)       ; STACK, AND PUTS THE RESULT
0916 33            XPAL  P3             ; INTO LO(P2) AND HI(P2)
0917 C410          LDI   H(AESTK)
0919 37            XPAH  P3
091A C300          LD    (P3)
091C CAEF          ST    LO(P2)
091E C301          LD    1(P3)
0920 CAEE          ST    HI(P2)
0922 90E5          JMP   X20

                   ;************************************
                   ;*             UNTIL                *
                   ;************************************

                   .LOCAL
0924 C2FF   UNTIL: LD    DOPTR(P2)      ;CHECK FOR DO-STACK UNDERFLOW
0926 01            XAE
0927 40            LDE
0928 E47A          XRI   L(DOSTAK)
092A 9C04          JNZ   $1
092C C40F          LDI   15
092E 90E0          JMP   E13
0930 C2EF   $1:    LD    LO(P2)         ;CHECK FOR EXPRESSION = 0
0932 DAEE          OR    HI(P2)
0934 9806          JZ    $REDO          ;IF ZERO, REPEAT DO-LOOP
0936 BAFF          DLD   DOPTR(P2)      ;ELSE POP SAVE STACK
0938 BAFF          DLD   DOPTR(P2)
093A 90CD          JMP   X20            ;CONTINUE TO NEXT STMT
093C 40     $REDO: LDE                  ;POINT P3 AT DO-STACK
093D 33            XPAL  P3
093E C410          LDI   H(DOSTAK)
0940 37            XPAH  P3
0941 C3FF          LD    -1(P3)         ;LOAD P1 FROM DO STACK
0943 35            XPAH  P1
0944 C3FE          LD    -2(P3)
0946 31            XPAL  P1             ;CURSOR NOW POINTS TO FIRST
0947 90C0          JMP   X20            ; STATEMENT OF DO-LOOP

                   ;************************************
                   ;*    STORE INTO STATUS REGISTER    *
                   ;************************************

                   ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
                   ;     'STAT' '=' REL-EXP

0949 C2EF   MOVESR: LD   LO(P2)         ;LOW BYTE GOES TO STATUS
094B D4F7          ANI   0F7            ; BUT WITH IEN BIT CLEARED
094D 07            CAS
094E 90B9   X21:   JMP   X20
0950 90BE   E14:   JMP   E13

                   ;************************************
                   ;*          STAT FUNCTION           *
                   ;************************************

0952 C410   STATUS: LDI   H(AESTK)
0954 37            XPAH  P3             ;POINT P3 AT AE STACK
0955 AAFD          ILD   LSTK(P2)
0957 AAFD          ILD   LSTK(P2)
0959 33            XPAL  P3
095A 06            CSA
095B CBFE          ST    -2(P3)         ;STATUS REG IS LOW BYTE
095D C400          LDI   0
095F CBFF          ST    -1(P3)         ;ZERO IS HIGH BYTE
0961 90EB          JMP   X21

                   ;************************************
                   ;*    MACHINE LANGUAGE SUBROUTINE   *
                   ;************************************

                   ; THIS ROUTINE IMPLEMENTS THE 'LINK' STATEMENT

0963 C2EE   CALLML: LD   HI(P2)         ;GET HIGH BYTE OF ADDRESS
0965 37            XPAH  P3
0966 C2EF          LD    LO(P2)         ;GET LOW BYTE
0968 33            XPAL  P3             ;P3 -> USER'S ROUTINE
0969 C7FF          LD    @-1(P3)        ;CORRECT P3
096B 3F            XPPC  P3             ;CALL ROUTINE (PRAY IT WORKS)
096C               LDPI  P2,VARS        ;RESTORE RAM POINTER
0972 90DA          JMP   X21            ;RETURN

                   ;************************************
                   ;*       SAVE DO LOOP ADDRESS       *
                   ;************************************

                   ; THIS ROUTINE IMPLEMENTS THE 'DO' STATEMENT.

                   .LOCAL
0974 C2FF   SAVEDO: LD   DOPTR(P2)      ;CHECK FOR STACK OVERFLOW
0976 E48A          XRI   L(FORSTK)
0978 9C04          JNZ   $1
097A C40A          LDI   10
097C 90D2   E15:   JMP   E14
097E AAFF   $1:    ILD   DOPTR(P2)
0980 AAFF          ILD   DOPTR(P2)
0982 33            XPAL  P3
0983 C410          LDI   H(DOSTAK)
0985 37            XPAH  P3             ;P3 -> TOP OF DO STACK
0986 35            XPAH  P1             ;SAVE CURSOR ON THE STACK
0987 CBFF          ST    -1(P3)
0989 35            XPAH  P1
098A 31            XPAL  P1
098B CBFE          ST    -2(P3)
098D 31            XPAL  P1
```

```
098E 90BE   X22:    JMP     X21


            ;*************************************
            ;*        TOP OF RAM FUNCTION        *
            ;*************************************

                    .LOCAL
0990 C2E9   TOP:    LD      TEMP2(P2)       ;SET P3 TO POINT TO
0992 37             XPAH    P3              ; START OF NIBL TEXT
0993 C2E8           LD      TEMP3(P2)
0995 33             XPAL    P3
0996 C300   $0:     LD      (P3)            ;HAVE WE HIT END OF TEXT?
0998 9402           JP      $1              ;NO - SKIP TO NEXT LINE
099A 9007           JMP     $2              ;YES - PUT CURSOR ON STACK
099C C302   $1:     LD      2(P3)           ;GET LENGTH OF LINE
099E 01             XAE
099F C780           LD      @EREG(P3)       ;SKIP TO NEXT LINE
09A1 90F3           JMP     $0              ;GO CHECK FOR EOF
09A3 C702   $2:     LD      @2(P3)          ;P3 := P3 + 2
09A5 AAFD           ILD     LSTK(P2)        ;SET P3 TO STACK, SAVING
09A7 AAFD           ILD     LSTK(P2)        ; OLD P3 (WHICH CONTAINS TOP)
09A9 33             XPAL    P3              ; ON IT SOMEHOW
09AA 01             XAE
09AB C410           LDI     H(AESTK)
09AD 37             XPAH    P3
09AE CBFF           ST      -1(P3)
09B0 40             LDE
09B1 CBFE           ST      -2(P3)
09B3 90D9           JMP     X22


            ;*************************************
            ;*      SKIP TO NEXT NIBL LINE       *
            ;*************************************

09B5 C501   IGNORE: LD      @1(P1)          ;SCAN TIL WE'RE PAST
09B7 E40D           XRI     0D              ; CARRIAGE RETURN
09B9 9CFA           JNZ     IGNORE
09BB 3F             XPPC    P3              ;YES - RETURN


            ;*************************************
            ;*          MODULO FUNCTION          *
            ;*************************************

09BC C2FD   MODULO: LD      LSTK(P2)        ;THIS ROUTINE MUST BE
09BE 33             XPAL    P3              ; IMMEDIATELY AFTER A
09BF C410           LDI     H(AESTK)        ; DIVIDE TO WORK CORRECTLY
09C1 37             XPAH    P3
09C2 C303           LD      3(P3)           ;GET LOW BYTE OF REMAINDER
09C4 CBFE           ST      -2(P3)          ;PUT ON STACK
09C6 C302           LD      2(P3)           ;GET HIGH BYTE OF REMAINDER
09C8 CBFF           ST      -1(P3)          ;PUT ON STACK
09CA 90C2   X23:    JMP     X22
09CC 90AE   E16:    JMP     E15


            ;*************************************
            ;*          RANDOM FUNCTION          *
            ;*************************************

                    .LOCAL
09CE C408   RANDOM: LDI     8               ;LOOP COUNTER FOR MULTIPLY
09D0 CAEB           ST      NUM(P2)
09D2 C2E5           LD      RNDX(P2)
09D4 01             XAE
09D5 C2E4           LD      RNDY(P2)
09D7 CAE9           ST      TEMP2(P2)
09D9 C2E5   $LOOP:  LD      RNDX(P2)        ;MULTIPLY THE SEEDS BY 9
09DB 02             CCL
09DC 70             ADE
09DD 01             XAE
09DE C2E4           LD      RNDY(P2)
09E0 02             CCL
09E1 F2E9           ADD     TEMP2(P2)
09E3 CAE4           ST      RNDY(P2)
09E5 BAEB           DLD     NUM(P2)
09E7 9CF0           JNZ     $LOOP
09E9 40             LDE                     ;ADD 7 TO SEEDS
09EA 02             CCL
09EB F407           ADI     7
09ED 01             XAE
09EE C2E4           LD      RNDY(P2)
09F0 02             CCL
09F1 F407           ADI     7
09F3 1E             RR
09F4 CAE4           ST      RNDY(P2)
09F6 AAE6           ILD     RNDF(P2)        ;HAVE WE GONE THROUGH
09F8 9803           JZ      $1              ; 256 GENERATIONS?
09FA 40             LDE                     ;IF SO, SKIP GENERATING
09FB CAE5           ST      RNDX(P2)        ; THE NEW RNDX
09FD C2FD   $1:     LD      LSTK(P2)        ;START MESSING WITH THE STACK
09FF 33             XPAL    P3
0A00 C410           LDI     H(AESTK)
0A02 37             XPAH    P3
0A03 C401           LDI     1               ;FIRST PUT 1 ON STACK
0A05 CB00           ST      (P3)
0A07 C400           LDI     0
0A09 CB01           ST      1(P3)
0A0B C3FE           LD      -2(P3)          ;PUT EXPR2 ON STACK
0A0D CB02           ST      2(P3)
0A0F C3FF           LD      -1(P3)
0A11 CB03           ST      3(P3)
0A13 C3FC           LD      -4(P3)          ;PUT EXPR1 ON STACK
0A15 CB04           ST      4(P3)
0A17 C3FD           LD      -3(P3)
0A19 CB05           ST      5(P3)
0A1B C2E4           LD      RNDY(P2)        ;PUT RANDOM # ON STACK
0A1D CBFE           ST      -2(P3)
0A1F C2E5           LD      RNDX(P2)
0A21 E4FF           XRI     0FF
0A23 D47F           ANI     07F
0A25 CBFF           ST      -1(P3)
0A27 C706           LD      @6(P3)          ;ADD 6 TO STACK POINTER
0A29 33             XPAL    P3
0A2A CAFD           ST      LSTK(P2)
0A2C 909C   X24:    JMP     X23
0A2E 909C   E16A:   JMP     E16


            ;*************************************
            ;*     PUSH 1 ON ARITHMETIC STACK    *
            ;*************************************

0A30 AAFD   LIT1:   ILD     LSTK(P2)
0A32 AAFD           ILD     LSTK(P2)
0A34 33             XPAL    P3
0A35 C410           LDI     H(AESTK)
0A37 37             XPAH    P3
0A38 C400           LDI     0
0A3A CBFF           ST      -1(P3)
0A3C C401           LDI     1
0A3E CBFE           ST      -2(P3)
0A40 90EA           JMP     X24


            ;*************************************
            ;*      FOR-LOOP INITIALIZATION      *
            ;*************************************

                    .LOCAL
0A42 C2FE   SAVFOR: LD      FORPTR(P2)      ;CHECK FOR FOR STACK
0A44 E4A6           XRI     L(PCSTAK)       ; OVERFLOW
0A46 9C04           JNZ     $1
0A48 C40A           LDI     10
0A4A 90E2   E17:    JMP     E16A
0A4C E4A6   $1:     XRI     L(PCSTAK)
0A4E 31             XPAL    P1              ;POINT P1 AT FOR STACK
0A4F CAF1           ST      P1LOW(P2)       ; SAVING OLD P1
0A51 C410           LDI     H(FORSTK)
0A53 35             XPAH    P1
0A54 CAF0           ST      P1HIGH(P2)
0A56 C2FD           LD      LSTK(P2)        ;POINT P3 AT AE STACK
0A58 33             XPAL    P3
0A59 C410           LDI     H(AESTK)
0A5B 37             XPAH    P3
0A5C C3F9           LD      -7(P3)          ;GET VARIABLE INDEX
0A5E CD01           ST      @1(P1)          ;SAVE ON FOR-STACK
0A60 C3FC           LD      -4(P3)          ;GET L(LIMIT)
0A62 CD01           ST      @1(P1)          ;SAVE
0A64 C3FD           LD      -3(P3)          ;GET H(LIMIT)*
0A66 CD01           ST      @1(P1)          ;SAVE
0A68 C3FE           LD      -2(P3)          ;GET L(STEP)
0A6A CD01           ST      @1(P1)          ;SAVE
0A6C C3FF           LD      -1(P3)          ;GET H(STEP)
0A6E CD01           ST      @1(P1)          ;SAVE
0A70 C2F1           LD      P1LOW(P2)       ;GET L(P1)
0A72 CD01           ST      @1(P1)          ;SAVE
0A74 C2F0           LD      P1HIGH(P2)      ;GET H(P1)
0A76 CD01           ST      @1(P1)          ;SAVE
0A78 35             XPAH    P1              ;RESTORE OLD P1
0A79 C2F1           LD      P1LOW(P2)
0A7B 31             XPAL    P1
0A7C CAFE           ST      FORPTR(P2)      ;UPDATE FOR STACK PTR
0A7E C7FC           LD      @-4(P3)
0A80 33             XPAL    P3
0A81 CAFD           ST      LSTK(P2)        ;UPDATE AE STACK PTR
0A83 90A7   X25:    JMP     X24


            ;*************************************
            ;*     FIRST PART OF 'NEXT VAR'      *
            ;*************************************

                    .LOCAL
0A85 C2FE   NEXTV:  LD      FORPTR(P2)      ;POINT P1 AT FOR STACK,
0A87 E48A           XRI     L(FORSTK)       ; CHECKING FOR UNDERFLOW
0A89 9C04           JNZ     $1
0A8B C40B           LDI     11              ;REPORT ERROR
0A8D 90BB           JMP     E17
0A8F E48A   $1:     XRI     L(FORSTK)
0A91 31             XPAL    P1
0A92 CAF1           ST      P1LOW(P2)       ;SAVE OLD P1
0A94 C410           LDI     H(FORSTK)
0A96 35             XPAH    P1
0A97 CAF0           ST      P1HIGH(P2)
0A99 C2FD           LD      LSTK(P2)        ;POINT P3 AT AE STACK
0A9B 33             XPAL    P3
0A9C C410           LDI     H(AESTK)
0A9E 37             XPAH    P3
0A9F C7FF           LD      @-1(P3)         ;GET VARIABLE INDEX
0AA1 E1F9           XOR     -7(P1)          ;COMPARE WITH INDEX
0AA3 9804           JZ      $10             ; ON FOR STACK: ERROR
0AA5 C40C           LDI     12              ; IF NOT EQUAL
0AA7 90A1   E18:    JMP     E17
0AA9 E1F9   $10:    XOR     -7(P1)          ;RESTORE INDEX
0AAB 01             XAE                     ;SAVE IN EREG
0AAC C280           LD      EREG(P2)        ;GET L(VARIABLE)
0AAE 02             CCL
0AAF F1FC           ADD     -4(P1)          ;ADD L(STEP)
0AB1 CA80           ST      EREG(P2)        ;STORE IN VARIABLE
0AB3 CB00           ST      (P3)            ; AND ON STACK
0AB5 C601           LD      @1(P2)          ;INCREMENT RAM PTR
0AB7 C280           LD      EREG(P2)        ;GET H(VARIABLE)
0AB9 F1FD           ADD     -3(P1)          ;ADD H(STEP)
0ABB CA80           ST      EREG(P2)        ;STORE IN VARIABLE
0ABD CB01           ST      1(P3)           ; AND ON STACK
0ABF C6FF           LD      @-1(P2)         ;RESTORE RAM POINTER
0AC1 C1FA           LD      -6(P1)          ;GET L(LIMIT)
0AC3 CB02           ST      2(P3)           ;PUT ON STACK
0AC5 C1FB           LD      -5(P1)          ;GET H(LIMIT)
0AC7 CB03           ST      3(P3)           ;PUT ON STACK
0AC9 C1FD           LD      -3(P1)          ;GET H(STEP)
0ACB 9410           JP      $2              ;IF NEGATIVE, INVERT
0ACD C404           LDI     4               ; ITEMS ON A.E. STACK
0ACF CAEB           ST      NUM(P2)         ;NUM = LOOP COUNTER
0AD1 C701   $LOOP:  LD      @1(P3)          ;GET BYTE FROM STACK
0AD3 E4FF           XRI     0FF             ;INVERT IT
0AD5 CBFF           ST      -1(P3)          ;PUT BACK ON STACK
0AD7 BAEB           DLD     NUM(P2)         ;DO UNTIL NUM = 0
0AD9 9CF6           JNZ     $LOOP
0ADB 9002           JMP     $3
0ADD C704   $2:     LD      @4(P3)          ;UPDATE AE STACK POINTER
0ADF 33     $3:     XPAL    P3
0AE0 CAFD           ST      LSTK(P2)
0AE2 C2F1           LD      P1LOW(P2)       ;RESTORE OLD P1
0AE4 31             XPAL    P1
0AE5 C2F0           LD      P1HIGH(P2)
0AE7 35             XPAH    P1
0AE8 9099   X26:    JMP     X25
```

```
                ;************************************
                ;*     SECOND PART OF 'NEXT VAR'    *
                ;************************************

0AEA C2EF   NEXTV1: LD      LO(P2)          ;IS FOR-LOOP OVER WITH?
0AEC 9808           JZ      $REDO           ;NO - REPEAT LOOP
0AEE C2FE           LD      FORPTR(P2)      ;YES - POP FOR-STACK
0AF0 02             CCL
0AF1 F4F9           ADI     -7
0AF3 CAFE           ST      FORPTR(P2)
0AF5 3F             XPPC    P3              ;RETURN TO I.L. INTERPRETER
0AF6 C2FE   $REDO:  LD      FORPTR(P2)      ;POINT P3 AT FOR STACK
0AF8 33             XPAL    P3
0AF9 C410           LDI     H(FORSTK)
0AFB 37             XPAH    P3
0AFC C3FF           LD      -1(P3)          ;GET OLD P1 OFF STACK
0AFE 35             XPAH    P1
0AFF C3FE           LD      -2(P3)
0B01 31             XPAL    P1
0B02 90E4           JMP     X26
0B04 90A1   E19:    JMP     E18
                ;************************************
                ;*     PRINT MEMORY AS STRING       *
                ;************************************

                ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
                ;     'PRINT' '$' FACTOR

                .LOCAL
0B06 C2EE   PSTRNG: LD      HI(P2)          ;POINT P1 AT STRING TO PRINT
0B08 35             XPAH    P1
0B09 C2EF           LD      LO(P2)
0B0B 31             XPAL    P1
0B0C                LDPI    P3,PUTC-1       ;POINT P3 AT PUTC ROUTINE
0B12 C501   $1:     LD      @1(P1)          ;GET A CHARACTER
0B14 E40D           XRI     0D              ;IS IT A CARRIAGE RETURN?
0B16 98D0           JZ      X26             ;YES - WE'RE DONE
0B18 E40D           XRI     0D              ;NO - PRINT THE CHARACTER
0B1A 3F             XPPC    P3
0B1B 06             CSA                     ;MAKE SURE NO ONE IS
0B1C D420           ANI     020             ; TYPING ON THE TTY
0B1E 9CF2           JNZ     $1              ; BEFORE REPEATING LOOP
0B20 90C6           JMP     X26

                ;************************************
                ;*        INPUT A STRING            *
                ;************************************

                ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
                ;     'INPUT' '$' FACTOR

0B22 C2EE   ISTRNG: LD      HI(P2)          ;GET ADDRESS TO STORE THE
0B24 37             XPAH    P3              ; STRING, PUT IT INTO P3
0B25 C2EF           LD      LO(P2)
0B27 33             XPAL    P3
0B28 C501   $2:     LD      @1(P1)          ;GET A BYTE FROM LINE BUFFER
0B2A CF01           ST      @1(P3)          ;PUT IT IN SPECIFIED LOCATION
0B2C E40D           XRI     0D              ;DO UNTIL CHAR = CARR. RETURN
0B2E 9CF8           JNZ     $2
0B30 90B6   X27:    JMP     X26

                ;************************************
                ;*   STRING CONSTANT ASSIGNMENT     *
                ;************************************

                ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
                ;     '$' FACTOR '=' STRING

                .LOCAL
0B32 C2EF   PUTSTR: LD      LO(P2)          ;GET ADDRESS TO STORE STRING,
0B34 33             XPAL    P3              ; PUT IT INTO P3
0B35 C2EE           LD      HI(P2)
0B37 37             XPAH    P3
0B38 C501   $LOOP:  LD      @1(P1)          ;GET A BYTE FROM STRING
0B3A E422           XRI     '"'             ;CHECK FOR END OF STRING
0B3C 980E           JZ      $END
0B3E E42F           XRI     '"' ! 0D        ;MAKE SURE THERE'S NO CR
0B40 9C04           JNZ     $1
0B42 C407           LDI     7
0B44 90BE           JMP     E19             ;ERROR IF CARRIAGE RETURN
0B46 E40D   $1:     XRI     0D              ;RESTORE CHARACTER
0B48 CF01           ST      @1(P3)          ;PUT IN SPECIFIED LOCATION
0B4A 90EC           JMP     $LOOP           ;GET NEXT CHARACTER
0B4C C40D   $END:   LDI     0D              ;APPEND CARRIAGE RETURN
0B4E CB00           ST      (P3)            ; TO STRING
0B50 90DE           JMP     X27

                ;************************************
                ;*        MOVE STRING               *
                ;************************************

                ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
                ;     '$' FACTOR '=' '$' FACTOR

                .LOCAL
0B52 C2FD   MOVSTR: LD      LSTK(P2)        ;POINT P3 AT A.E. STACK
0B54 33             XPAL    P3
0B55 C410           LDI     H(AESTK)
0B57 37             XPAH    P3
0B58 C7FF           LD      @-1(P3)         ;GET ADDRESS OF SOURCE STRING
0B5A 35             XPAH    P1              ; INTO P1
0B5B C7FF           LD      @-1(P3)
0B5D 31             XPAL    P1
0B5E C7FF           LD      @-1(P3)         ;GET ADDRESS OF DESTINATION
0B60 01             XAE                     ; STRING INTO P3
0B61 C7FF           LD      @-1(P3)
0B63 33             XPAL    P3
0B64 CAFD           ST      LSTK(P2)        ;UPDATE STACK POINTER
0B66 40             LDE
0B67 37             XPAH    P3
0B68 C501   $LOOP:  LD      @1(P1)          ;GET A SOURCE CHARACTER
0B6A CF01           ST      @1(P3)          ;SEND IT TO DESTINATION
0B6C E40D           XRI     0D              ;REPEAT UNTIL CARRIAGE RET.
0B6E 98C0           JZ      X27
0B70 06             CSA                     ; OR KEYBOARD INTERRUPT
0B71 D420           ANI     020
0B73 9CF3           JNZ     $LOOP
0B75 90B9           JMP     X27

                ;************************************
                ;*     PUT PAGE NUMBER ON STACK     *
                ;************************************

0B77 AAFD   PUTPGE: ILD     LSTK(P2)
0B79 AAFD           ILD     LSTK(P2)
0B7B 33             XPAL    P3
0B7C C410           LDI     H(AESTK)
0B7E 37             XPAH    P3
0B7F C2F6           LD      PAGE(P2)
0B81 CBFE           ST      -2(P3)
0B83 C400           LDI     0
0B85 CBFF           ST      -1(P3)
0B87 90A7           JMP     X27

                ;************************************
                ;*        ASSIGN NEW PAGE           *
                ;************************************

                .LOCAL
0B89 C2EF   NUPAGE: LD      LO(P2)          ;GET PAGE # FROM STACK,
0B8B D407           ANI     7               ; GET THE LOW 3 BITS
0B8D 9C02           JNZ     $0              ;PAGE 0 BECOMES PAGE 1
0B8F C401           LDI     1
0B91 CAF6   $0:     ST      PAGE(P2)
0B93 3F             XPPC    P3              ;RETURN

                ;************************************
                ;*        FIND START OF PAGE        *
                ;************************************

                ; THIS ROUTINE COMPUTES THE START OF THE CURRENT TEXT PAGE,
                ; STORING THE ADDRESS IN TEMP2(P2) (THE HIGH BYTE), AND
                ; TEMP3(P2) (THE LOW BYTE).

0B94 C2F6   FNDPGE: LD      PAGE(P2)
0B96 E401           XRI     1               ;SPECIAL CASE IS PAGE 1, BUT
0B98 9C09           JNZ     $1              ; OTHERS ARE CONVENTIONAL
0B9A C411           LDI     H(PGM)          ;PAGE 1 STARTS AT 'PGM'
0B9C CAE9           ST      TEMP2(P2)
0B9E C420           LDI     L(PGM)
0BA0 CAE8           ST      TEMP3(P2)
0BA2 3F             XPPC    P3              ;RETURN
0BA3 E401   $1:     XRI     1               ;RESTORE PAGE #
0BA5 01             XAE                     ;SAVE IT
0BA6 C404           LDI     4               ;LOOP COUNTER = 4
0BA8 CAEB           ST      NUM(P2)
0BAA 40     $LOOP:  LDE                     ;MULTIPLY PAGE# BY 16
0BAB 02             CCL
0BAC 70             ADE
0BAD 01             XAE
0BAE BAEB           DLD     NUM(P2)
0BB0 9CF8           JNZ     $LOOP
0BB2 40             LDE
0BB3 CAE9           ST      TEMP2(P2)       ;TEMP2 HAS HIGH BYTE
0BB5 C402           LDI     2               ; OF ADDRESS NOW
0BB7 CAE8           ST      TEMP3(P2)       ;LOW BYTE IS ALWAYS 2
0BB9 3F             XPPC    P3
                ;************************************
                ;*      MOVE CURSOR TO NEW PAGE     *
                ;************************************

0BBA C2E9   CHPAGE: LD      TEMP2(P2)       ;PUT START OF PAGE
0BBC 35             XPAH    P1              ; INTO P1.  THIS ROUTINE
0BBD C2E8           LD      TEMP3(P2)       ; MUST BE CALLED RIGHT
0BBF 31             XPAL    P1              ; AFTER 'FNDPGE'
0BC0 3F             XPPC    P3              ;RETURN

                ;************************************
                ;*      DETERMINE CURRENT PAGE      *
                ;************************************

0BC1 35     DETPGE: XPAH    P1              ;CURRENT PAGE IS HIGH
0BC2 01             XAE                     ; PART OF CURSOR DIVIDED
0BC3 40             LDE                     ; BY 16
0BC4 35             XPAH    P1
0BC5 40             LDE
0BC6 1C             SR
0BC7 1C             SR
0BC8 1C             SR
0BC9 1C             SR
0BCA CAF6           ST      PAGE(P2)
0BCC 3F             XPPC    P3              ;RETURN

                ;************************************
                ;*       CLEAR CURRENT PAGE         *
                ;************************************

0BCD C2E9   NEWPGM: LD      TEMP2(P2)       ;POINT P1 AT CURRENT PAGE
0BCF 35             XPAH    P1
0BD0 C2E8           LD      TEMP3(P2)
0BD2 31             XPAL    P1
0BD3 C40D           LDI     0D              ;PUT DUMMY END-OF-LINE
0BD5 C9FF           ST      -1(P1)          ; JUST BEFORE TEXT
0BD7 C4FF           LDI     -1              ;PUT -1 AT START OF TEXT
0BD9 C900           ST      (P1)
0BDB C901           ST      1(P1)
0BDD 3F             XPPC    P3              ;RETURN

                ;************************************
                ;*      FIND LINE NUMBER IN TEXT    *
                ;************************************

                ;INPUTS:  THE START OF THE CURRENT PAGE IN TEMP2 AND TEMP3,
                ;         THE LINE NUMBER TO LOOK FOR IN LO AND HI.
                ;OUPUTS:  THE ADDRESS OF THE FIRST LINE IN THE NIBL TEXT
                ;         WHOSE LINE NUMBER IS GREATER THAN OR EQUAL TO THE
                ;         NUMBER IN HI AND LO, RETURNED IN ADRLO AND ADRHI.

                .LOCAL
0BDE C2E9   FNDLBL: LD      TEMP2(P2)       ;POINT P1 AT START OF TEXT
0BE0 35             XPAH    P1
```

```
0BE1 C2E8           LD      TEMP3(P2)
0BE3 31             XPAL    P1
0BE4 C100   $1:     LD      (P1)            ;HAVE WE HIT END OF TEXT?
0BE6 E4FF           XRI     0FF
0BE8 9412           JP      $2              ;YES - STOP LOOKING
0BEA 03             SCL                     ;NO - COMPARE LINE NUMBERS
0BEB C101           LD      1(P1)           ; BY SUBTRACTING
0BED FAEF           CAD     LO(P2)
0BEF C100           LD      0(P1)
0BF1 FAEE           CAD     HI(P2)          ;IS TEXT LINE # >= LINE #?
0BF3 9407           JP      $2              ;YES - STOP LOOKING
0BF5 C102           LD      2(P1)           ;NO - TRY NEXT LINE IN TEXT
0BF7 01             XAE
0BF8 C580           LD      @EREG(P1)       ; SKIP LENGTH OF LINE
0BFA 90E8           JMP     $1
0BFC 31     $2:     XPAL    P1              ;SAVE ADDRESS OF FOUND LINE
0BFD CAF3           ST      LABLLO(P2)      ; IN LABLHI AND LABLLO
0BFF 31             XPAL    P1
0C00 35             XPAH    P1
0C01 CAF2           ST      LABLHI(P2)
0C03 35             XPAH    P1
0C04 C2EF           LD      LO(P2)          ;WAS THERE AN EXACT MATCH?
0C06 E101           XOR     1(P1)
0C08 9C07           JNZ     $3
0C0A C2EE           LD      HI(P2)
0C0C E100           XOR     0(P1)
0C0E 9C01           JNZ     $3              ;NO - FLAG THE ADDRESS
0C10 3F             XPPC    P3              ;YES - RETURN NORMALLY
0C11 C2F2   $3:     LD      LABLHI(P2)      ;SET SIGN BIT OF HIGH PART
0C13 DC80           ORI     080             ; OF ADDRESS TO INDICATE
0C15 CAF2           ST      LABLHI(P2)      ; INEXACT MATCH OF LINE #'S
0C17 3F             XPPC    P3
                    .PAGE   ' I. L. MACROS'

                ;***********************************
                ;*          I. L. MACROS           *
                ;***********************************

                    .LOCAL

    2000    $TSTBIT =       TSTBIT*256
    8000    $CALBIT =       CALBIT*256
    4000    $JMPBIT =       JMPBIT*256

                    .MACRO  TST,FAIL,A,B
                    .DBYTE  $TSTBIT!FAIL
                    .IF     #=2
                    .BYTE   'A'!080
                    .ELSE
                    .ASCII  'A'
                    .BYTE   'B'!080
                    .ENDIF
                    .ENDM

                    .MACRO  TSTCR,FAIL
                    .DBYTE  $TSTBIT!FAIL
                    .BYTE   0D!080
                    .ENDM

                    .MACRO  TSTV,FAIL
                    .ADDR   TSTVAR
                    .DBYTE  FAIL
                    .ENDM

                    .MACRO  TSTN,FAIL
                    .ADDR   TSTNUM
                    .DBYTE  FAIL
                    .ENDM

                    .MACRO  JUMP,ADR
                    .DBYTE  $JMPBIT!ADR
                    .ENDM

                    .MACRO  CALL,ADR
                    .DBYTE  $CALBIT!ADR
                    .ENDM

                    .MACRO  DO
                    .MLOC   I
                    .SET    I,1
                    .DO     #
                    .ADDR   #I
                    .SET    I,I+1
                    .ENDDO
                    .ENDM

                    .PAGE   ' I. L. TABLE'

                ;***********************************
                ;*            I. L. TABLE          *
                ;***********************************

0C18        START:  DO      NLINE
0C1A        PROMPT: DO      GETL
0C1C                TSTCR   PRMPT1
0C1F                JUMP    PROMPT
0C21        PRMPT1: TSTN    LIST
0C25                DO      FNDPGE,XCHGP1,POPAE,FNDLBL,INSRT
0C2F                JUMP    PROMPT

0C31        LIST:   TST     RUN,'LIS','T'
0C37                DO      FNDPGE
0C39                TSTN    LIST1
0C3D                DO      POPAE,FNDLBL
0C41                JUMP    LIST2
0C43        LIST1:  DO      CHPAGE
0C45        LIST2:  DO      LST
0C47        LIST3:  CALL    PRNUM
0C49                DO      LST3
0C4B                JUMP    START

0C4D        RUN:    TST     CLR,'RU','N'
0C52                DO      DONE
0C54        BEGIN:  DO      FNDPGE,CHPAGE,STRT,NXT

0C5C        CLR:    TST     NEW,'CLEA','R'
0C63                DO      DONE,CLEAR,NXT

0C69        NEW:    TST     STMT,'NE','W'
0C6E                TSTN    DFAULT

0C72                JUMP    NEW1
0C74        DFAULT: DO      LIT1
0C76        NEW1:   DO      DONE,POPAE,NUPAGE,FNDPGE,NEWPGM,NXT

0C82        STMT:   TST     LET,'LE','T'
0C87        LET:    TSTV    AT
0C8B                TST     SYNTAX,'='
0C8E                CALL    RELEXP
0C90                DO      STORE,DONE,NXT
0C96        AT:     TST     IF,'@'
0C99                CALL    FACTOR
0C9B                TST     SYNTAX,'='
0C9E                CALL    RELEXP
0CA0                DO      MOVE,DONE,NXT

0CA6        IF:     TST     UNT,'I','F'
0CAA                CALL    RELEXP
0CAC                TST     IF1,'THE','N'
0CB2        IF1:    DO      POPAE,CMPR
0CB6                JUMP    STMT
0CB8        UNT:    TST     DO,'UNTI','L'
0CBF                DO      CKMODE
0CC1                CALL    RELEXP
0CC3                DO      DONE,POPAE,UNTIL,DETPGE,NXT

0CCD        DO:     TST     GOTO,'D','O'
0CD1                DO      CKMODE,DONE,SAVEDO,NXT

0CD9        GOTO:   TST     RETURN,'G','O'
0CDD                TST     GOSUB,'T','O'
0CE1                CALL    RELEXP
0CE3                DO      DONE
0CE5                JUMP    GO1
0CE7        GOSUB:  TST     SYNTAX,'SU','B'
0CEC                CALL    RELEXP
0CEE                DO      DONE,SAV
0CF2        GO1:    DO      FNDPGE,POPAE,FNDLBL,XFER,NXT

0CFC        RETURN: TST     NEXT,'RETUR','N'
0D04                DO      DONE,RSTR,DETPGE,NXT

0D0C        NEXT:   TST     FOR,'NEX','T'
0D12                DO      CKMODE
0D14                TSTV    SYNTAX
0D18                DO      DONE,NEXTV
0D1C                CALL    GTROP
0D1E                DO      POPAE,NEXTV1,DETPGE,NXT

0D26        FOR:    TST     STAT,'FO','R'
0D2B                DO      CKMODE
0D2D                TSTV    SYNTAX
0D31                TST     SYNTAX,'='
0D34                CALL    RELEXP
0D36                TST     SYNTAX,'T','O'
0D3A                CALL    RELEXP
0D3C                TST     FOR1,'STE','P'
0D42                CALL    RELEXP
0D44                JUMP    FOR2
0D46        FOR1:   DO      LIT1
0D48        FOR2:   DO      DONE,SAVFOR,STORE,NXT

0D50        STAT:   TST     PGE,'STA','T'
0D56                TST     SYNTAX,'='
0D59                CALL    RELEXP
0D5B                DO      POPAE,MOVESR
0D5F                DO      DONE,NXT

0D63        PGE:    TST     DOLLAR,'PAG','E'
0D69                TST     SYNTAX,'='
0D6C                CALL    RELEXP
0D6E                DO      DONE,POPAE,NUPAGE,FNDPGE,CHPAGE,NXT

0D7A        DOLLAR: TST     PRINT,'$'
0D7D                CALL    FACTOR
0D7F                TST     SYNTAX,'='
0D82                TST     DOLR1,'"'
0D85                DO      POPAE,PUTSTR
0D89                JUMP    DOLR2
0D8B        DOLR1:  TST     SYNTAX,'$'
0D8E                CALL    FACTOR
0D90                DO      XCHGP1,MOVSTR,XCHGP1
0D96        DOLR2:  DO      DONE,NXT

0D9A        PRINT:  TST     INPUT,'P','R'
0D9E                TST     PR1,'IN','T'
0DA3        PR1:    TST     PR2,'"'
0DA6                DO      PRS
0DA8                JUMP    COMMA
0DAA        PR2:    TST     PR3,'$'
0DAD                CALL    FACTOR
0DAF                DO      XCHGP1,POPAE,PSTRNG,XCHGP1
0DB7                JUMP    COMMA
0DB9        PR3:    CALL    RELEXP
0DBB                CALL    PRNUM
0DBD        COMMA:  TST     PR4,','
0DC0                JUMP    PR1
0DC2        PR4:    TST     PR5,';'
0DC5                JUMP    PR6
0DC7        PR5:    DO      NLINE
0DC9        PR6:    DO      DONE,NXT

0DCD        INPUT:  TST     END,'INPU','T'
0DD4                DO      CKMODE
0DD6                TSTV    IN2
0DDA                DO      XCHGP1,GETL
0DDE        IN1:    CALL    RELEXP
0DE0                DO      STORE,XCHGP1
0DE4                TST     IN3,','
0DE7                TSTV    SYNTAX
0DEB                DO      XCHGP1
0DED                TST     SYNTAX,','
0DF0                JUMP    IN1
0DF2        IN2:    TST     SYNTAX,'$'
0DF5                CALL    FACTOR
0DF7                DO      XCHGP1,GETL,POPAE,ISTRNG,XCHGP1
0E01        IN3:    DO      DONE,NXT

0E05        END:    TST     ML,'EN','D'
0E0A                DO      DONE,BREAK
```

```
0E0E        ML:     TST     REM,'LIN','K'
0E14                CALL    RELEXP
0E16                DO      DONE,XCHGP1,POPAE,CALLML,XCHGP1,NXT

0E22        REM:    TST     SYNTAX,'RE','M'
0E27                DO      IGNORE,NXT
0E2B        SYNTAX: DO      ERR
0E2D        ERRNUM: CALL    PRNUM
0E2F                DO      FIN

                ; NOTE: EACH RELATIONAL OPERATOR (EQ, LEQ, ETC.) DOES AN
                ; AUTOMATIC 'RTN' (THIS SAVES VALUABE BYTES!)

0E31        RELEXP: CALL    EXPR
0E33                TST     REL1,'='
0E36                CALL    EXPR
0E38                DO      EQ
0E3A        REL1:   TST     REL4,'<'
0E3D                TST     REL2,'='
0E40                CALL    EXPR
0E42                DO      LEQ
0E44        REL2:   TST     REL3,'>'
0E47                CALL    EXPR
0E49                DO      NEQ
0E4B        REL3:   CALL    EXPR
0E4D                DO      LSS
0E4F        REL4:   TST     RETEXP,'>'
0E52                TST     REL5,'='
0E55                CALL    EXPR
0E57                DO      GEQ
0E59        REL5:   CALL    EXPR
0E5B        GTROP:  DO      GTR

0E5D        EXPR:   TST     EX1,'-'
0E60                CALL    TERM
0E62                DO      NEG
0E64                JUMP    EX3
0E66        EX1:    TST     EX2,'+'
0E69        EX2:    CALL    TERM
0E6B        EX3:    TST     EX4,'+'
0E6E                CALL    TERM
0E70                DO      ADD
0E72                JUMP    EX3
0E74        EX4:    TST     EX5,'-'
0E77                CALL    TERM
0E79                DO      SUB
0E7B                JUMP    EX3
0E7D        EX5:    TST     RETEXP,'O','R'
0E81                CALL    TERM
0E83                DO      OROP
0E85                JUMP    EX3
0E87        RETEXP: DO      RTN

0E89        TERM:   CALL    FACTOR
0E8B        T1:     TST     T2,'*'
0E8E                CALL    FACTOR
0E90                DO      MUL
0E92                JUMP    T1
0E94        T2:     TST     T3,'/'
0E97                CALL    FACTOR
0E99                DO      DIV
0E9B                JUMP    T1
0E9D        T3:     TST     RETEXP,'AN','D'
0EA2                CALL    FACTOR
0EA4                DO      ANDOP
0EA6                JUMP    T1

0EA8        FACTOR: TSTV    F1
0EAC                DO      IND,RTN
0EB0        F1:     TSTN    F2
0EB4                DO      RTN
0EB6        F2:     TST     F3,'#'
0EB9                DO      HEX,RTN
0EBD        F3:     TST     F4,'('
0EC0                CALL    RELEXP
0EC2                TST     SYNTAX,')'
0EC5                DO      RTN
0EC7        F4:     TST     F5,'@'
0ECA                CALL    FACTOR
0ECC                DO      EVAL,RTN
0ED0        F5:     TST     F6,'NO','T'
0ED5                CALL    FACTOR
0ED7                DO      NOTOP,RTN
0EDB        F6:     TST     F7,'STA','T'
0EE1                DO      STATUS,RTN
0EE5        F7:     TST     F8,'TO','P'
0EEA                DO      FNDPGE,TOP,RTN
0EF0        F8:     TST     F9,'MO','D'
0EF5                CALL    DOUBLE
0EF7                DO      DIV,MODULO,RTN
0EFD        F9:     TST     F10,'RN','D'
0F02                CALL    DOUBLE
0F04                DO      RANDOM,SUB,ADD,DIV,MODULO,ADD,RTN
0F12        F10:    TST     SYNTAX,'PAG','E'
0F18                DO      PUTPGE,RTN

0F1C        DOUBLE: TST     SYNTAX,'('
0F1F                CALL    RELEXP
0F21                TST     SYNTAX,','
0F24                CALL    RELEXP
0F26                TST     SYNTAX,')'
0F29                DO      RTN

0F2B        PRNUM:  DO      XCHGP1,PRN
0F2F        PRNUM1: DO      DIV,PRN1,XCHGP1,RTN
                    PAGE    'ERROR MESSAGES'

                ;************************************
                ;*          ERROR MESSAGES          *
                ;************************************

                        .MACRO  MESSAGE,A,B
                        .ASCII  'A'
                        .BYTE   'B'!080
                        .ENDM

0F37        MESGS:  MESSAGE ' ERRO','R'     ; 1
0F3D                MESSAGE 'ARE','A'       ; 2
0F41                MESSAGE 'STM','T'       ; 3
0F45                MESSAGE 'CHA','R'       ; 4
0F49                MESSAGE 'SNT','X'       ; 5
0F4D                MESSAGE 'VAL','U'       ; 6
0F51                MESSAGE 'END','"'       ; 7
0F55                MESSAGE 'NOG','O'       ; 8
0F59                MESSAGE 'RTR','N'       ; 9
0F5D                MESSAGE 'NES','T'       ; 10
0F61                MESSAGE 'NEX','T'       ; 11
0F65                MESSAGE 'FO','R'        ; 12
0F68                MESSAGE 'DIV','O'       ; 13
0F6C                MESSAGE 'BR','K'        ; 14
0F6F                MESSAGE 'UNT','L'       ; 15
                    PAGE    ' TELETYPE ROUTINES'

                ;************************************
                ;*     GET CHARACTER AND ECHO IT    *
                ;************************************
                ;
                        .LOCAL
0F73 C408   GECO:   LDI     8               ;SET COUNT = 8
0F75 CAEB           ST      NUM(P2)
0F77 06             CSA                     ;SET READER RELAY
0F78 DC02           ORI     2
0F7A 07             CAS
0F7B 06     $1:     CSA                     ;WAIT FOR START BIT
0F7C D420           ANI     020
0F7E 9CFB           JNZ     $1              ;NOT FOUND
0F80 C457           LDI     87              ;DELAY 1/2 BIT TIME
0F82 8F04           DLY     4
0F84 06             CSA                     ;IS START BIT STILL THERE?
0F85 D420           ANI     020
0F87 9CF2           JNZ     $1              ;NO
0F89 06             CSA                     ;SEND START BIT
0F8A D4FD           ANI     %2              ;RESET READER RELAY
0F8C DC01           ORI     1
0F8E 07             CAS
0F8F C485   $2:     LDI     133             ;DELAY 1 BIT TIME
0F91 8F08           DLY     8
0F93 06             CSA                     ;GET BIT (SENSEB)
0F94 D420           ANI     020
0F96 9804           JZ      $3
0F98 C401           LDI     1
0F9A 9004           JMP     $4
0F9C C400   $3:     LDI     0
0F9E 9C00           JNZ     $4
0FA0 CAEA   $4:     ST      TEMP(P2)        ;SAVE BIT VALUE (0 OR 1)
0FA2 1F             RRL                     ;ROTATE INTO LINK
0FA3 01             XAE
0FA4 1D             SRL                     ;SHIFT INTO CHARACTER
0FA5 01             XAE                     ;RETURN CHAR TO E
0FA6 06             CSA                     ;ECHO BIT TO OUTPUT
0FA7 DC01           ORI     1
0FA9 E2EA           XOR     TEMP(P2)
0FAB 07             CAS
0FAC BAEB           DLD     NUM(P2)         ;DECREMENT BIT COUNT
0FAE 9CDF           JNZ     $2              ;LOOP UNTIL 0
0FB0 06             CSA                     ;SET STOP BIT
0FB1 D4FE           ANI     0FE
0FB3 07             CAS
0FB4 8F08           DLY     8               ;DELAY APPROX. 2 BIT TIMES
0FB6 40             LDE                     ;AC HAS INPUT CHARACTER
0FB7 D47F           ANI     07F
0FB9 01             XAE
0FBA 40             LDE
0FBB 3F             XPPC    P3              ;RETURN
0FBC 90B5           JMP     GECO
                ;************************************
                ;*     PRINT CHARACTER AT TTY       *
                ;************************************

0FBE 01     PUTC:   XAE
0FBF C4FF           LDI     255
0FC1 8F17           DLY     23
0FC3 06             CSA                     ;SET OUTPUT BIT TO LOGIC 0
0FC4 DC01           ORI     1               ; FOR START BIT. (NOTE INVERS
0FC6 07             CAS
0FC7 C409           LDI     9               ;INITIALIZE BIT COUNT
0FC9 CAE8           ST      TEMP3(P2)
0FCB C48A   PUTC1:  LDI     138             ;DELAY 1 BIT TIME
0FCD 8F08           DLY     8
0FCF BAE8           DLD     TEMP3(P2)       ;DECREMENT BIT COUNT.
0FD1 9810           JZ      PUTC2
0FD3 40             LDE                     ;PREPARE NEXT BIT
0FD4 D401           ANI     1
0FD6 CAE9           ST      TEMP2(P2)
0FD8 01             XAE                     ;SHIFT DATA RIGHT 1 BIT
0FD9 1C             SR
0FDA 01             XAE
0FDB 06             CSA                     ;SET UP OUTPUT BIT
0FDC DC01           ORI     1
0FDE E2E9           XOR     TEMP2(P2)
0FE0 07             CAS                     ;PUT BIT TO TTY
0FE1 90E8           JMP     PUTC1
0FE3 06     PUTC2:  CSA                     ;SET STOP BIT
0FE4 D4FE           ANI     0FE
0FE6 07             CAS
0FE7 3F             XPPC    P3              ;RETURN
0FE8 90D4           JMP     PUTC
0000                .END    0
```

ADD	0335	AESTK	1050	ANDOP	05F0	AT	0C96
BEGIN	0C54	BREAK	0288	CALBIT	0080	CALLML	0963
CHEAT	007C	CHEAT1	009B	CHPAGE	0BBA	CHRNUM	FFE7
CK1	0649	CKMODE	0644	CLEAR	0051	CLEAR1	0056
CLR	0C5C	CMP	0562	CMP1	05C2	CMP2	05CA
CMPR	05D9	COMMA	0DBD	DETPGE	0BC1	DFAULT	0C74
DIV	0410	DO	0CCD	DOLLAR	0D7A	DOLR1	0D8B
DOLR2	0D96	DONE	0135	DONE1	0143	DONE2	0144
DOPTR	FFFF	DOSTAK	107A	DOUBLE	0F1C	E0	0150
E0A	010D	E1	0195	E10	07C6	E11	081E
E12	086D	E12A	08E1	E13	0910	E14	0950
E15	097C	E16	09CC	E16A	0A2E	E17	0A4A
E18	0AA7	E19	0B04	E2	01CC	E3A	028A
E4	02DF	E5	030C	E6	0378	E6A	03D2
E8	064B	E8A	06E5	E8B	06B2	E9	074F
END	0E05	EQ	054C	EREG	FF80	ERR	0223
ERR1	0225	ERR2	0227	ERRNUM	0E2D	EVAL	07E6
EX1	0E66	EX2	0E69	EX3	0E6B	EX4	0E74
EX5	0E7D	EXECIL	0076	EXPR	0E5D	F1	0EB0
F10	0F12	F2	0EB6	F3	0EBD	F4	0EC7
F5	0ED0	F6	0EDB	F7	0EE5	F8	0EF0
F9	0EFD	FACTOR	0EA8	FAIL	05E1	FAILHI	FFEC

```
FAILLO FFED     FALSE  05C8     FIN    02B2     FNDLBL 0BDE
FNDPGE 0B94     FOR    0D26     FOR1   0D46     FOR2   0D48
FORPTR FFFE     FORSTK 108A     GECO   0F73     GEQ    0560
GEQ1   05BE     GETL   0753     GO1    0CF2     GOSUB  0CE7
GOTO   0CD9     GTR    055C     GTR1   05B3     GTROP  0E5B
HEX    0654     HI     FFEE     HILINE FFF7     IF     0CA6
IF1    0CB2     IGNORE 09B5     ILC1   00AA     ILCALL 00A0
IN1    0DDE     IN2    0DF2     IN3    0E01     IND    0534
INPUT  0DCD     INSRT  0820     ISTRNG 0B22     JMPBIT 0040
LABLHI FFF2     LABLLO FFF3     LBUF   10D6     LEQ    0558
LEQ1   05AA     LET    0C87     LIST   0C31     LIST1  0C43
LIST2  0C45     LIST3  0C47     LISTNG FFF5     LIT1   0A30
LO     FFEF     LOLINE FFF8     LSS    0554     LSS1   05A2
LST    02E1     LST2   02FF     LST3   030E     LST4   0314
LST5   0324     LSTK   FFFD     MESGS  0F37     ML     0E0E
MODULO 09BC     MOVE   0804     MOVESR 0949     MOVSTR 0B52
MUL    037A     NEG    0363     NEQ    0550     NEQ1   0599
NEW    0C69     NEW1   0C76     NEWPGM 0BCD     NEXT   0D0C
NEXTV  0A85     NEXTV1 0AEA     NLINE  0215     NOJUMP 009D
NOTOP  05F8     NUM    FFEB     NUPAGE 0B89     NXT    028C
NXT1   02A9     OROP   05F4     P1     0001     P1HIGH FFF0
P1LOW  FFF1     P2     0002     P3     0003     PAGE   FFF6
PCHIGH FFFA     PCLOW  FFFB     PCSTAK 10A6     PCSTK  FFF9
PGE    0D63     PGM    1120     POPAE  0912     PR1    0DA3
PR2    0DAA     PR3    0DB9     PR4    0DC2     PR5    0DC7
PR6    0DC9     PRINT  0D9A     PRMPT1 0C21     PRN    0197
PRN1   01CE     PRNUM  0F2B     PRNUM1 0F2F     PROMPT 0C1A
PRS    017E     PRS1   0193     PSTRNG 0B06     PUTC   0FBE
PUTC1  0FCB     PUTC2  0FE3     PUTPGE 0B77     PUTSTR 0B32
RANDOM 09CE     REL1   0E3A     REL2   0E44     REL3   0E4B
REL4   0E4F     REL5   0E59     RELEXP 0E31     REM    0E22
RETEXP 0E87     RETURN 0CFC     RNDF   FFE6     RNDX   FFE5
RNDY   FFE4     RSTR   0148     RSTR1  0152     RSTR2  0167
RTN    00FB     RUN    0C4D     RUNMOD FFF4     SAV    010F
SAV1   012B     SAV2   0131     SAVEDO 0974     SAVFOR 0A42
SBRPTR FFFC     SBRSTK 106A     SETZ   058D     START  0C18
STAT   0D50     STATUS 0952     STMT   0C82     STORE  04C9
STRT   02C8     SUB    034C     SYNTAX 0E2B     T1     0E8B
T2     0E94     T3     0E9D     TEMP   FFEA     TEMP2  FFE9
TEMP3  FFE8     TERM   0E89     TOP    0990     TST    00C5
TSTBIT 0020     TSTNUM 06B4     TSTVAR 04E9     UNT    0CB8
UNTIL  0924     VARS   101C     X0     00EC     X1     0165
X10    04E2     X11    054A     X12    0597     X12A   05EE
X12B   0637     X12C   067A     X13    06E3     X14    0751
X15    07B4     X16    07E4     X17    081C     X19    086B
X19A   08E5     X20    0909     X21    094E     X22    098E
X23    09CA     X24    0A2C     X25    0A83     X26    0AE8
X27    0B30     X4     01CA     X5     0221     X5A    0286
X6     02DD     X6A    030A     X7     034A     X8     0376
X9     03EF     X9A    0439     X9B    04A4     XCHGP1 0639
XFER   0171     XFER1  0179     ZZ0001 101C     ZZ0002 1120
ZZ0003 0FBD     ZZ0004 0FBD     ZZ0005 0FBD     ZZ0006 0FBD
ZZ0007 0F37     ZZ0008 0F37     ZZ0009 0FBD     ZZ000A 10D6
ZZ000B 0FBD     ZZ000C 101C     ZZ000D 0FBD     ZZ000E 0002
ZZ000F 0002     ZZ0010 0006     ZZ0011 0002     ZZ0012 0003
ZZ0013 0002     ZZ0014 0002     ZZ0015 0002     ZZ0016 0002
ZZ0017 0005     ZZ0018 0004     ZZ0019 0002     ZZ001A 0007
ZZ001B 0004     ZZ001C 0004     ZZ001D 0003     ZZ001E 0002
ZZ001F 0006     ZZ0020 0005     ZZ0021 0002     ZZ0022 0003
ZZ0023 0006     ZZ0024 0005     ZZ0025 0002     ZZ0026 0003
ZZ0027 0005     ZZ0028 0002     ZZ0029 0002     ZZ002A 0005
ZZ002B 0003     ZZ002C 0003     ZZ002D 0007     ZZ002E 0003
ZZ002F 0004     ZZ0030 0003     ZZ0031 0002     ZZ0032 0005
ZZ0033 0002     ZZ0034 0003     ZZ0035 0002     ZZ0036 0003
ZZ0037 0003     ZZ0038 0002     ZZ0039 0006     ZZ003A 0003
ZZ003B 0003     ZZ003C 0007     ZZ003D 0003     ZZ003E 0002
ZZ003F 0002     ZZ0040 0002     ZZ0041 0002     ZZ0042 0002
ZZ0043 0002     ZZ0044 0002     ZZ0045 0002     ZZ0046 0002
ZZ0047 0002     ZZ0048 0002     ZZ0049 0002     ZZ004A 0002
ZZ004B 0002     ZZ004C 0002     ZZ004D 0002     ZZ004E 0003
ZZ004F 0002     ZZ0050 0003     ZZ0051 0002     ZZ0052 0003
ZZ0053 0003     ZZ0054 0003     ZZ0055 0004     ZZ0056 0004
ZZ0057 0008     ZZ0058 0003     ZZ0059 0002     ZZ005A 0003
ZZ005B 0005     $0     002A     $0     0420     $0     076F
$0     0996     $0     0B91     $1     0043     $1     01C6
$1     023D     $1     0397     $1     0443     $1     05FA
$1     06E7     $1     0772     $1     0838     $1     0930
$1     097E     $1     099C     $1     09FD     $1     0A4C
$1     0A8F     $1     0B12     $1     0B46     $1     0BA3
$1     0BE4     $1     0F7B     $10    0AA9     $2     01FF
$2     025A     $2     03A8     $2     0454     $2     0707
$2     07B2 *   $2     0842     $2     09A3     $2     0ADD
$2     0B28     $2     0BFC     $2     0F8F     $3     0261
$3     03D4     $3     04AC     $3     084A     $3     0ADF
$3     0C11     $3     0F9C     $4     03F1     $4     0857
$4     0FA0     $5     085E     $ABOR  06C6     $ADD   08E3
$ADD1  08FF     $CALB  8000     $CR    07D8     $DOWN  0885
$END   04C3     $END   06AC     $END   0B4C     $ENT1  04A6
$ENTE  068B     $ENTE  07B6     $EXIT  0402     $FAIL  04FB
$JMPB  4000     $LETR  067C     $LOOP  002C     $LOOP  00DB
$LOOP  0203     $LOOP  0241     $LOOP  03B6     $LOOP  0460
$LOOP  066C     $LOOP  06F7     $LOOP  09D9     $LOOP  0AD1
$LOOP  0B38     $LOOP  0B68     $LOOP  0BAA     $MAYB  050D
$MOVE  086F     $MSG   0247     $NEQ   00EE     $NOT   0628
$OK    051A     $OK    0688     $OR    0616     $POS   043B
$PRNT  01EF     $REDO  093C     $REDO  0AF6     $RET   06D6
$RUB   07CE     $SCAN  00C7     $SHIF  0692     $SHIF  0714
$SKIP  0664     $TSTB  2000     $UP    0895     $UP1   08A1
$UP2   08C2     $UP3   08C4     $UP4   08D4     $XH    07C8
$XU    07A6

NO ERROR LINES
SOURCE CHECKSUM = 33FE
INPUT FILE    1:NIBL2.SRC
```

6502 STRING OUTPUT, REVISITED

Dear Mr. Warren, Oct. 6, 1976

In *DDJ*, Vol. 1, No. 8 (p. 33), Mr. Espinosa proposed the exchange of "handy" subroutines to save bytes in space-limited systems. He also presented an example, an ASCII string output subroutine for the 6502 microprocessor. I would like to submit a revised version of Mr. Espinosa's subroutine. I have done extensive work on 6502's with OSI's Model 400 microcomputer. During this time I have learned several byte saving programming "tricks" which I would like to pass on by illustration. Through a few simple changes I was able to reduce the length from 40 to 2B (hex) bytes. The result is a subroutine which works the same and saves a few more bytes. The program demonstrates a few simple "tricks":

- Preservation of the Y index register on the stack (3 bytes saved)
- Replace JMP instruction (with ranges less than 128 bt bytes) with forced relative branches. This permits easier relocation of a generalized subroutine so it may be used elsewhere in memory.
- Make use of TYA instruction rather than saving the Y index in a memory location and then adding it in later (5 bytes saved).
- Test the carry flag condition and increment the high order byte if set rather than adding 00 (2 bytes saved).
- Try to avoid dead space inside programs, and non-sero page data storage (i.e. locations 0433 to 043F) (12 bytes saved).

Sincereiy,
Marcel Meier

8850 S. Spring Valley Dr.
Chagrin Falls, OH 44022

```
                ; STRINGOUT: REVISED VERSION
                ; ORIGINAL BY C. ESPINOSA
                ; REVISIONS BY M. MEIER
                ; 10/5/76
                ;
                        ORG $400
                AKEEP   EQU $42A
                OUT     EQU $FEEE
                LO      EQU $EE
                HI      EQU $FF
                LO      DP7
                HI      DP7
                ;
0400  8D 2A 04  BEGIN   STA AKEEP       SAVE AC
0403  68                PLA             GET RETURN ADDRESS
0404  85 EE             STA LO
0406  68                PLA
0407  85 FF             STA HI
0409  98                TYA             PUT Y INDEX ON STACK
040A  48                PHA
040B  A0 01             LDY #01         SET UP INDEX POINTER
040D  B1 FF       NEXT  LDA (LO),Y      GET NEXT CHARACTER
040F  F0 07             BEQ EXIT        DONE IF NULL CHARACTER
0411  C8                INY
0412  20 FF FF          JSR OUT         OUTPUT CHARACTER
0415  18                CLC             FORCED LOOP WITH
0416  90 F5             BCC NEXT        A RELATIVE BRANCH
0418  98          EXIT  TYA             GET STRING LENGTH
0419  38                SEC
041A  65 EE             ADC LO          ADD RETURN ADDRESS TO
041C  85 EE             STA LO          OFFSET
041E  90 02             BCC NOCAR       IF CARRY, INCREMENT HI
0420  E6 FF             INC HI
0422  68          NOCAR PLA             RESTORE Y FROM STACK
0423  A8                TAY
0424  AD 2A 04          LDA AKEEP       RESTORE AC
0427  6C EE 00          JMP (LO)        RETURN TO INSTRUCTION AFTER NULL
```

IN-GROUP HUMOR FOR DINOSAUR USERS

We recently heard of some new instructions proposed for some of the maxi computers of industry and business:

BRANCH & BOMB
BRANCH & HANG
PUNCH OPERATOR
BACKSPACE & EJECT DISC
BACKSPACE & PUNCH DISC

Oh well; we *said* it was in-group humor.

TSC LIVES! THEY *DO* HAVE A PHONE NUMBER

Technical Systems Consultants, Box 2574, West Lafayette, IN 47906, peddles some interesting, low-cost micro software. Several people have asked us if TSC is OK to deal with, stating that they were unable to locate a phone number or street address. We wish to emphatically state that they *are* real; they *are* reputable; and they *do* have a phone: (317) 742-7509.

UPGRADED CP/M FLOPPY DISC OPERATING SYSTEM NOW AVAILABLE

CP/M is a disk operating system designed for diskette-based computer systems which use the Intel 8080 microcomputer. The CP/M software package is now being offered to the small computer user community.

Previously available only to OEM's, CP/M has been in existence for over two years in various manufacturers' products, and thus has had extensive field testing. CP/M functions include file management, with console interaction, batch processing, and program loading facilities. The overall operation of CP/M closely resembles the standard features of the DEC System-10. In particular, CP/M components include:

BDOS – the CP/M Basic Disk Operating System supports a named file system, with up to 64 distinct files on each diskette. Files storage is dynamically allocated and released as necessary, with algorithms for optimal read/write head movement. Any file can contain as few as zero bytes, and up to 250K bytes, depending upon the requirements of the user program. Sequential and random access are supported.

CCP – the Console Command Processor interacts with the programmer's console, providing the basic commands:

DIR selectively search the disk directory for files
TYPE type the contents of a file at the console
REN rename a specific file to a different name
ERA erase a given file or set of files from the disk
SAVE save memory on the disk for later reload or test

The CCP also supports automatic program load and execution of CP/M system programs as well as user programs.

PIP – the CP/M Peripheral Interchange Program allows transfer of files between various devices and disk files, as well as concatenation of files on the diskettes.

SUBMIT – the batch processing features of CP/M allow the operator to prepare command files with parametric substitution, which can be subsequently automatically executed if typed by the operator.

ED – the CP/M editor allows preparation of programs and text using powerful context editing and display commands.

ASM – the CP/M assembler is compatible with both the standard Intel assembler and Processor Technology assembly language.

DDT – the CP/M Debugging Tool is a monitor which allows symbolic program tracing, debugging, and testing.

LOAD – the loader prepared a "memory image" file from an Intel format "hex" file, ready for direct execution under CP/M.

DUMP – the dump utility prints the contents of a CP/M file in hexadecimal at the user's console.

SYSGEN – the system generation utility prints the contents of a CP/M system diskettes from existing diskettes for back-up purposes.

The CP/M operating system is distributed for an Intel MDS microcomputer development system, but can be easily altered to operate with a wide variety of customized hardware environments. Basic requirements are:

a) Intel 8080 – based microcomputer mainframe
b) At least 16K of read/write main memory
c) One or two IBM-compatible disk drives and controller

Given these facilities, the CP/M disk system is "patched" by the user to communicate with the specialized hardware. The exact steps to follow in programming and patching the CP/M system are given in the manual *CP/M System Alteration Guide.* In fact, several popular mainframe and controller manufacturers currently support their own CP/M patch.

The CP/M system is distributed on an IBM-compatible diskette in machine-code form only (source programs are available for internal use, or distribution with custom hardware at additional cost), along with complete documentation required for operating CP/M and programming in the CP/M environment. The software is licensed for use by the individual who purchases CP/M, and is registered and serialized to prevent unauthorized copying and distribution. In particular, the licensing agreement specifically disallows copying CP/M for use by any individual other than the registered owner. The registered owner of a CP/M system receives notices of updates and becomes a member of the CP/M User's Library. System documentation includes:

CP/M Features and Facilities – this manual presents the organization of the CP/M system, along with the forms for file name references, built-in commands and transient commands, including operation of the editor, assembler, debugger, peripheral interchange program, and batch processor.

CP/M Editor, CP/M Assembler and *CP/M Debugger Manuals* – these three manuals provide the operating details for CP/M's principal subsystems for program composition (ED), assembly (ASM), and testing (DDT). Manuals can be purchased separately.

CP/M Interface Guide – this manual gives the exact details for programming in the CP/M environment. In particular, all system calls are specified, along with details of CP/M file organization which is necessary for programs which operate upon CP/M files.

CP/M System Alteration Guide – the alteration guide gives the step-by-step process which you must follow in order to alter the CP/M system to run with non-standard hardware. I/O drivers for commonly available hardware systems are given.

Individual manuals are $5. A package consisting of all six manuals is $25. An initialized, "loaded" floppy disc is $50. A disc and all documentation – "the works" – is $70. And, of course, Californians get to add 6% tax.

Digital Research, Box 579, Pacific Grove, CA 93950, (408) 373-3403.

PRAISE FOR DIGITAL SYSTEMS' FLOPPY UNITS & DIGITAL RESEARCH'S CP/M

Dear Dr. Dobb, Nov. 1, 1976

I have seen the articles on the CP/M floppy disc operating system available from Digital Systems. I am writing because I am a satisfied customer. I have had a system from Digital Systems running for nearly a year now and have had no trouble with it. The hardware is reliable and well designed. I do not know of anything presently on the market that compares favorably to it. The software is also fantastic and reliable. It is easy to interface with the DOS to read and write files, and do I/O. The software developed by Digital Research is well designed and is implemented much like the Monitor on the DEC System 10. The assembler, editor and debugger supported by the system are excellent. In addition to that the documentation that comes with the system is first class. I am enthusiastically pleased with the performance of the system.

I have dealt with Digital Systems and can unqualifiedly say that they are honest, decent and responsive. Dr. Torode was exceptionally helpful in getting the system up and supporting me afterwards. I have not encountered a more honest and responsive vendor.

The software written by Digital Research is excellent in design and documentation and to me it would be worth five times the price.

Altogether the combination of hardware and software which is provided turns an 8080 system into a true software development system which is flexible, easy to use, easy to learn, and reliable.

Sincerely,

Robert Swartz
195 Ivy Lane
Highland Park, IL 60035

A *SUPER*, TURNKEY DUAL FLOPPY SYSTEM

Dear Jim, Sept. 17, 1976

You guys are usually way ahead on new products and things but have you seem the DTC Micro File?

It's a WOW!

- 8080A Super System
- Has an extremely high quality, compact, dual floppy
- Has superb system software including fantastic text editor
- Uses MITS BASIC (they bought it) plus numerous improvements
- Speeds to 9600 baud through two RS323 ports

It might appear as a commercial system to you folks (it is!)–you should check it against IMSAI's dual disk system. It runs rings around them on price and is far superior. Price - $4295.

If you haven't seen it you should take a look.

Keep up the excellent work with *DDJ.*

Sincerely,

A. C. Delmas
ADVANCE SYSTEMS
P. O. Box 531
Saratoga, CA 95070

[We heard identical remarks from another friend whose judgment has been impeccable. DTC is located at 1190 Dell Ave., Bldg. L, Campbell, CA 95008, (408) 378-1112.–Editor]

```
RUN

BTRANS  13-SEP-76  BASIC V01B-02

        **** BASIC ARITHMETIC EXPRESSION TRANSLATOR ****
EXPR.?A=5
#### A = 5 ####
EXPR.?Z=6
#### Z = 6 ####
EXPR.?D=4
#### D = 4 ####
EXPR.?X=3.25
#### X = 3.25 ####
EXPR.?P=3.14159
#### P = 3.14159 ####
EXPR.?A*A
#### A*A = 25 ####
EXPR.?A/Z*P
#### A/Z*P = .265258 ####
EXPR.?(A*Z)+D*X
#### (A*Z)+D*X = 43 ####
EXPR.?((A-Z)/(X*Z)+P
((A-Z)/(X*Z)+P
MISSING )
EXPR.?((A-Z)/(X*Z)+P)
#### ((A-Z)/(X*Z)+P) = 3.09031 ####
EXPR.?STOP

STOP AT LINE 250

READY
```

```
BTRANS  13-SEP-76  BASIC V01B-02

10 REM          ***************  BTRANS  **********************
20 PRINT '      **** BASIC ARITHMETIC EXPRESSION TRANSLATOR ****
30 REM          FROM: DR DOBB'S JOUR. OF COMP. CALISTH. & ORTH.
40 REM          JUNE/JULY 1976 PP. 11-12        BY BILL THOMPSON
50 REM                  MODIFIED BY JBA 3-SEP-76
55 REM
60 REM          INIT ARRAYS & TRANS. TABLE
70 GOSUB 1000
72 REM
75 REM          LOOP THROUGH INTERPRETING INSTRUCTIONS
78 FOR N9=1 TO 1000
79 N8=0
80 REM          INIT STACK
90 GOSUB 1500
100 REM         START OF INPUT LOOP
110 PRINT 'EXPR.';\INPUT A$
115 IF A$='STOP' THEN 250
120 IF SEG$(A$,2,2)<>'=' THEN 160
125 REM         EXECUTE ASSIGN. STAT.
130 GOSUB 2000
140 GO TO 240
150 REM
160 REM         EVALUATE EXPRESSION
170 FOR K=1 TO 72
180 L=POS(B$,SEG$(A$,K,K),1)
190 IF L=0 THEN 210 \GOSUB 3000 \GO TO 220
210 GOSUB 2500
220 IF N8<>0 THEN 240
230 NEXT K
240 NEXT N9
250 STOP
1000 REM
1005 REM                        S/R TO INIT TRANS. TABLE & ARRAYS
1010 DIM C(26),T(80),E(80),D(6,7)
1012 REM                SET UP TRANS TABLE
1015 FOR I=1 TO 6\FOR J=1 TO 7
1020 READ D(I,J)
1030 NEXT J\NEXT I
1040 REM                TRANSLATION TABLE DATA
1045 REM  ( + - * / )
1050 DATA 7,1,1,1,1,1,6
1060 DATA 5,1,1,1,1,1,3
1070 DATA 4,1,2,2,1,1,4
1080 DATA 4,1,2,2,1,1,4
1090 DATA 4,1,4,4,1,1,4
1100 DATA 4,1,4,4,1,1,4
1110 REM
1120 REM                INIT VAR. STORAGE & STRINGS
1130 FOR I=1 TO 26
1140 C(I)=0
1150 NEXT I
1160 B$=' (+-*/)'
1170 C$='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
1175 RETURN
1500 REM
1510 REM                        S/R TO INITIALIZE STACK
1515 IF N8<1 THEN 1520 \RETURN
1520 T8=80\E8=80
1530 T(80)=1\E(80)=0
1540 RETURN
```

ARITHMETIC EXPRESSION EVALUATOR MOD

Sept. 13, 1976

Gentlemen:

Enclosed you will find a modification of Bill Thompson's arithmetic expression evaluator published in your June/July 1976 edition. As another unintiated person on the subject of interpreters, I found his BASIC program to be enlightening.

However, after trying out his program, I did find that his stack operations could be handled much more efficiently by moving pointers to the stack tops, rather than the entire stacks. Also, I tried to make the flow a little more readable by limiting the use of GOTO statements to within the same subroutine and by starting the line numbers of the subroutines in increments of 500. Since my mod was implemented on a PDP-11, there are some minor differences in the string functions.

Thanks,

Jim Abshire

508 - 4th St.
Laurel, MD 28810

```
2000 REM
2010 REM                              S/R FOR ASSIGNMENT STATEMENT
2015 IF N8<1 THEN 2020 \RETURN
2020 M=POS(C$,SEG$(A$,1,1),1)
2030 N=POS(A$,'=',1)
2040 C(M)=VAL(SEG$(A$,N+1,N+10))
2050 P$='#### '&SEG$(C$,M,M)&' = '&STR$(C(M))&' ####'
2060 PRINT P$
2070 RETURN
2500 REM
2505 REM                              S/R TO PUT VALUE ON E STACK TOP
2510 IF N8<1 THEN 2515 \RETURN
2515 M=POS(C$,SEG$(A$,K,K),1)
2520 IF M=0 THEN 2570
2525 E8=E8-1
2530 IF E8<2 THEN 2550 \E(E8)=C(M)
2540 GO TO 2610
2550 PRINT 'E STACK OVERFLOW'\N8=1\GO TO 2610
2570 REM
2580 REM                    ERROR HANDLING
2590 GOSUB 5500 \PRINT 'INVALID SYMBOL'\N8=1
2610 RETURN
3000 REM
3010 REM                              S/R TO EXECUTE INSTRUCTIONS
3015 IF N8<1 THEN 3020 \RETURN
3020 D=D(T(T8),L)
3030 IF D=7 THEN 3380
3040 IF D=6 THEN 3330
3050 IF D=5 THEN 3280
3060 IF D=4 THEN 3230
3070 IF D=3 THEN 3180
3080 IF D=2 THEN 3130
3090 REM
3100 REM                    INSTR. 1
3110 GOSUB 3500
3120 GO TO 3440
3130 REM
3140 REM                    INSTR. 2
3150 GOSUB 4000
3160 GOSUB 3500
3170 GO TO 3440
3180 REM
3190 REM                    INSTR. 3
3200 GOSUB 4500
3210 GO TO 3440
3220 REM
3230 REM                    INSTR. 4
3240 GOSUB 4000
3250 GOSUB 4500
3260 GOSUB 3000
3270 GO TO 3440
3280 REM
3290 REM                    INSTR. 5
3300 GOSUB 5500
3310 PRINT 'MISSING )'
3320 N8=1\GO TO 3440
3330 REM
3340 REM                    INSTR. 6
3350 GOSUB 5500
3360 PRINT 'MISSING ('
3370 N8=1\GO TO 3440
3380 REM
3390 REM                    INSTR. 7
3400 P$='#### '&A$&' = '&STR$(E(E8))&' ####'
3410 PRINT P$
3420 GOSUB 5000
3430 E(E8)=0\N8=2
3440 RETURN
3500 REM
3510 REM                              S/R TO ADD SYMBOL TO T STACK
3515 IF N8<1 THEN 3520 \RETURN
3520 T8=T8-1
3530 IF T8<2 THEN 3550 \T(T8)=L\GO TO 3560
3550 PRINT 'T STACK OVERFLOW'\N8=1
3560 RETURN
4000 REM
4010 REM                              S/R TO GENERATE AN OPERATION
4015 IF N8<1 THEN 4020 \RETURN
4020 T9=T(T8)
4030 IF T9=7 THEN 4060 \IF T9=6 THEN 4240
4040 IF T9=5 THEN 4190 \IF T9=4 THEN 4140
4050 IF T9=3 THEN 4090
4060 GOSUB 5500
4070 PRINT 'OPER. GENERATOR ERROR'\N8=1\GO TO 4280
4090 REM
4100 REM                    ADD OPER.
4110 E(E8+1)=E(E8+1)+E(E8)
4120 GO TO 4270
4140 REM
4150 REM                    SUBTR. OPER.
4160 E(E8+1)=E(E8+1)-E(E8)
4170 GO TO 4270
4190 REM
4200 REM                    MULT. OPER.
4210 E(E8+1)=E(E8+1)*E(E8)
4220 GO TO 4270
4240 REM
4250 REM                    DIV. OPER.
4260 E(E8+1)=E(E8+1)/E(E8)
4270 GOSUB 5000
4280 RETURN
4500 REM
4510 REM                              S/R TO POP STACK T
4515 IF N8<1 THEN 4520 \RETURN
4520 T8=T8+1
4530 IF T8<=80 THEN 4540 \PRINT 'T STACK UNDERFLOW'\N8=1
4540 RETURN
5000 REM
5010 REM                              S/R TO POP STACK E
5015 IF N8<1 THEN 5020 \RETURN
5020 E8=E8+1
5030 IF E8<=80 THEN 5040 \PRINT 'E STACK UNDERFLOW'\N8=1
5040 RETURN
5500 REM
5510 REM                              S/R TO PRINT INPUT STRING
5515 IF N8<1 THEN 5520 \RETURN
5520 PRINT A$
5530 RETURN
6000 REM
6010 REM                              END OF PROGRAM
6020 END

READY
```

COMPONENTS FOR SPECIFYING PROGRAMMING LANGUAGES AND MODS TO THE *TINY HI* LANGUAGE DESIGN

Dear *DDJ,* Nov. 17, 1976

Enclosed are about a dozen changes in TINY HI, an updated language summary, and a brief description of HI. None of the changes significantly effect the scope of TINY, but I believe they make it an even nicer language. I am shelving TINY LISP, TINY SNOBOL, and the extensible language I mentioned; my system is up and I want to implement TINY. I renege on the promise to describe FORTH as *Interface* has had a good article on it.

I will act as a clearinghouse to standardize TINY HI implementations. A complete language standard (as detailed below) should be out by 15 January. Implementors please send $3 to cover copying costs and first class postage. This will be the last revision in *DDJ,* but I believe I've finalized what the user sees. *DDJ* will get a free copy of those standards for existing; otherwise I would have to invent it and couldn't do nearly as well. I hope to be HI in 77.

Laissez faire,

Martin Buchanan
(703) 893-7978
2040 Lord Fairfax Rd.
Vienna, VA 22180

ELEMENTS OF PROGRAMMING LANGUAGE STANDARDS

1. A complete semantic and syntactic description (mostly accomplished). This includes little things like the significance of blanks, levels of nesting, algorithms used for real or mixed arithmetic (in languages with real numbers), and identification of lexical tokens;
2. Storage formats for source programs, object programs, and data, both in main memory or on mass storage units;
3. Conventions for the naming and semantics of global functions or variables which handle hardware differences (.DEVICE, .MAINSIZE, etc.);
4. Standard names and algorithms for common library functions;
5. All error messages, when they are invoked, and their meaning;
6. Text-editing functions during data entry;
7. Interfacing with machine language programs;
8. Linking loader design;
9. Dynamic storage allocation and file retrieval design.

CHANGES TO TINY HI

Comments: a semicolon (";") in column one reserves only the line on which it appears for comments. A semicolon in any other column reserves that column and all to the right of it for comments until ";" is again encountered in the comment field. This replaces the "/*" and "*/" delimiters previously specified. The change makes commenting easier to learn and use, and increases flexibility.

Vectors: Vectors may have lengths up to $2^{32}-1$. Otherwise it would be almost impossible to handle data structures with more than 256 elements. This also allows any positive integer as a subscript.

Logical operators:AND, OR, NOT ; NOT is evaluated first. AND and OR have equal precedence. All three must be set off by blanks or)(as in:

```
NOT A=B      or   (A> MIN)AND(A<MAX)
```

Being able to express complex predicates is important in making structure as clear as possible, else many unnecessary IFs and DOs would make programs more confusing.

Continuation lines: a plus sign ("+") as the first nonblank character indicates a continuation line. A line may be continued indefinitely and even when comments intervene or there is a comment field. Continuation lines can be used to make output lists or complex predicates clearer by arranging them on several lines to show structure, and they also reduce the use of temporary variables.

Control structures: The "UNTIL" structure is now:

```
DO
  code
END IF p
```

The "WHILE" structure is now:

```
DO IF p
  code
END
```

The compound structure is also allowed:

```
DO IF p
  code
END IF q
```

Noise may no longer be added to END statements.

Professor Howard Tompkins of Indiana University of Pennsylvania caused my reexamination of my control structures, for which I am grateful, though we still disagree as to the best iterative structure. He pointed out that "UNTIL" should be "WHILE NOT" from the English meanings of the words, and also that UNTIL in COBOL has a meaning different from the one used by I and IBM. The new construct locates each predicate where it is actually examined, allows a new structure, reduces my vocabulary, eliminates a source of possible confusion, and allows for future integration with an iterative form:

```
DO I=J TO K BY L IF A(I) <A(I+1)
```

Input: ? alone will get a literal from the keyboard with the prompt "?←". ? followed by a variable will generate a prompt of the form: "<variable> ←"

Subvectors: In a sequence of numbers or characters, one often wants to indicate a subvector that is a continuous sequence, often a very long one. Other programming languages use pseudovariables (PL/I's SUBSTR function), "index generation" (APL), or novel subscripting forms (A[5;8] in HP BASIC 3000). My subscripting form for indicating subvectors should be familiar to any user of English. I call it "ellipsis". It is formed by three consecutive periods between the initial and final subscript, but separated by blanks from them (to avoid ambiguities when I introduce real numbers in HI):

```
A[4 . . . 11]  is the same as  A[4 5 6 7 8 9 10 11],
```

but both in conception and the generation of object code, the first is preferable.

Global indication: the "." prefix can be omitted from calls of external functions unless the function name is duplicated by a local fucntion. My theory is that data is usually local and functions are usually global.

Subscripting: is an operation and may apply to expressions:

```
(A+B)[2 3 5]
```

– MORE –

TINY HI, cont.

Arithmetic: append **, exponentiation. HI has it, and I want the differences between the two levels to be *few* and *major*. Exponentiation is also easy to implement in integer arithmetic. I prefer ** to the up arrow. Exponentiation derives from multiplication just as multiplication from addition, so the symbol is logical in some sense, and also common. I want to reserve the up arrow for a (presently undefined) sorting or ordering operation.

REVISED TINY LANGUAGE SUMMARY

Vocabulary: BEGIN END IF ELSE DO
Comments: ;
Continuation: +
Infix arithmetic: + – * / **
Prefix arithmetic: –
Concatenation: blank
Length operator: # "number"
Relational ops: < <= > >= <>
Logical ops: AND OR NOT
Assignment: ←
Input: ?
Global: .
Nesting: ( )
Subscripting: [ ]
Substring: . . . "ellipsis"
Data types: INTEGER STRING
Data structure: the vector

WHAT HI ADDS TO TINY HI

1. Data types REAL and LOGICAL, and the corresponding literals;
2. Multidimensional arrays;
3. Data declarations;
4. For program correctness, the attributes INITIAL, RANGE, and TYPE, and the ability to test an expression's type;
5. For output: the attribute FORMAT;
   functions SKIP, X, T;
   globals .COL, .LINES, .SPACE;
6. The iterative DO TO { BY }

POSTSCRIPT: Nov. 23, 1976

1. Negation of -2^{15} will produce an overflow.
2. Concatenation has a lower priority than # or negation, but still greater than the infix arithmetic operators. The example given for "number" should be "#(5 73 -1)".
3. Functions may have no argument, as in "CPTIME( )".
4. After "END WHILE" in GCF there should be the line "GCF ← Y".

I want to thank those who wrote about TINY HI, especially Gregg Townsend.

RCA 1802 PLEA

To: Jim Warren Nov. 2, 1976

I've not seen anything yet on the 1802. Is it too new for the hobbyist, or what? Could you publish a short request for responses from any 1802 users? [Yup!]

Sincerely,

Harley Shanko
15025 Vanowen St., No. 209
Van Nuys, CA 91405

TINY HI SUGGESTIONS

Dear Mr. Buchanan:

I would like to offer some suggestions regarding your TINY HI language as defined in the October, 1976, *Dr. Dobb's Journal*.

First of all, let me say that I LIKE IT! It seems to be quite powerful in its simplicity.

I like the one-statement-per line format; PL/I addicts look down their noses at FORTRAN for this but readable programs require it. I like being able to easily put comments on the same lines as statements; this is what often makes the *comments* of assembly-language programs better than those of so-called higher level languages. "?" for input is great, and I admire the simplicity of the vector scheme.

Now for the comments: (this is more or less random order)

1. Negation of -2^{15} will probably produce overflow; so perhaps there is a case where negation can produce an error.
2. Rather than bracketing loops with WHILE ... END, UNTIL ... END, how about using LOOP and REPEAT?
   a. WHILE and UNTIL do not imply iteration except to a programmer who's seen them before.
   b. WHILE *cond* terminates when *cond* goes false, and UNTIL *cond* terminates when it goes true; but it's very hard to see why one of these implies a test at the front of the loop and the other at the back.

   This isn't original – see Knuth in *Computing Surveys* Vol. 4 No. 6 (Dec. 1974), pp. 278-280.

```
LOOP IF cond          LOOP
       .                .
       .                .
     code             code
       .                .
       .                .
   REPEAT          REPEAT IF cond

(test at top)     (test at bottom)
```

3. How about providing a means for the n + ½loop problem? Again stealing from Knuth:

```
LOOP                  LOOP
"ENTER A" ?A          "ENTER A" ?A
WHILE A >47 for       IF A >47 EXITLOOP
"TOO BIG"             "TOO BIG"
REPEAT                REPEAT
```

4. I finally figured out why the example "No. 5 73-1" looks strange to me: because I can't get used to a unary operator with a higher precedence than a binary operator ($). All unary operators (#?-) should be higher than the binary operators. Under the current rules, (-5 7 9) = (-5) (-7) (-9)!
5. What determines whether input is taken as string or integer? How can 123 be input as a string?
6. What sets the value of a function? Should the example have an additional line .GCF ← Y?
7. The current syntax disallows null arguments such as .CPTIME(). Is this intentional?
8. Deletion of the /* or */ line has the potential for causing a lot of trouble when editing a program. I would favor a scheme such as it is used by some assemblers where ; means that everything else on the line is a comment.
9. Since a string is really a vector of characters, and you allow vectors of strings, will you allow vectors of vectors (of vectors . . .)?
10. I agree with ← for assignment but please choose the character to be used with ASCii keyboards before every implementor picks a different one.

I guess that's all that comes to mind now. I'm sending a copy of this to *DDJ*. Keep up the good work!

Yours,
Gregg Townsend

450 N. Mathilda, No. J20
Sunnyvale, CA 94086
Nov. 15, 1976

NEW COMPUTER MART

The Computer Mart of New Hampshire is currently located on Daniel Webster Hwy N, Merrimack, NH 03054, (603) 424-2981. On January 1st, it will move to 170 Main St., Nashua, NH 03060. [information from Ron Cordova, 76-12-4]

6800 MONITOR RELATIONS

Dear Editor, Nov. 10, 1976

Motorola makes several monitor roms (Mikbug*, Minibug*, Minibug II* and Exbug*) for their M6800 systems. Most systems in hobbyist hands are currently using Mikbug*. Minibug II* now seems to be available from Mini Micro-Mart and it has several additional and enhanced features over Mikbug*. These features are serial I/O to an ACIA for the control interface, binary load, binary dump, S9 on last record of punch, user control of "SWI" vector, upward and downward movement during address changes and memory test commands. Documentation on the commands is supplied, but no listing or hardward implementation guides. We are using Minibug II* in a SWTP 6800 and are pleased with its operation. The following notes are supplied for those who might wish to try this rom.

Dennis Sutherland
2835 - 25th Ave.
Marion, IA

David Kyllingstad
840 Hillview Dr.
Marion, IA

Ron Tonneson
Fairfax, IA

*Trademark of Motorola

MINIBUG II SOFTWARE EQUIVALENCE

Minibug II is not to be confused with Minibug which is located in the upper half of Mikbug and is probably not worth finding at this point in time. The following entry points have been tested and appear to work in programs that reference them.

ROUTINE	MIKBUG ADDRESS	MINIBUG II ADDRESS
OUTCH	E1D1	E108
INCH	E1AC	E11F
OUTHL	E067	E0FA
OUTHR	E06B	E0FE
OUTS	E0CC	E180
PDATA1	E07E	E130
CONTRL	E0E3	E040
INHEX	E0AA	E070
BADDR	E047	E0D9
OUT4HS	E0C8	E17C
OUT2HS	E0CA	E17E

MINIBUG II HARDWARE CHANGES REQUIRED

General

Since Minibug II is a IK rom, A_9 must be made active instead of being grounded. CS_0 and CS_1 are active low instead of active high as was Mikbug. This requires inverting the logic pins 10 and 11.

SWTPc

Isolate IC2 pin 15 (A_9) from the large ground buss by making a cut around the plated through that now connects pin 15 to ground. Do not drill out the through. Use a miniature circular saw, a miniature fly cutter or an Exacto knife. Now connect the isolated pad (IC2-15) to the pad immediately to the right which comes from IC-13.

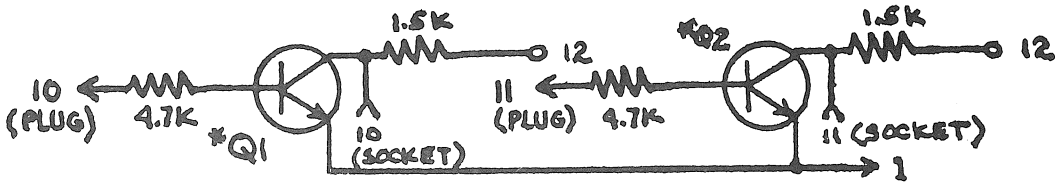
If a semi permanent change is anticipated, cut the lines coming from IC2-10 and 11 just past the first bend. Connect a jumper from IC16-8 to IC2-10 (CS_0). Connect a jumper from IC13-4 to IC2-11 (CS_1).

If plug-in interchangeability is desired, don't make the last two cuts but add two inverters.

The inverters may be made and installed between a 24 pin IC plug and a 24 pin socket (available from James Electronics). See schematic below.

Mount the socket piggyback on the plug and solder all other pins one to one. (Pins 1-9 and 12-24 are straight through, pins 10 and 11 are now inverted).

*Any plastic NPN switching transistor (2N5210, MPS3646, etc.)



GLITCH: TINY BASIC & MEK SYSTEMS

Dear *DDJ*, Oct. 26, 1976

I was referred a copy of what appears to be a column in the CHG-NT newsletter, which briefly mentions a failure of Tiny BASIC 6800 in Mot Eval kits.

It is true that I have had a number of calls from owners of MEK systems in which Tiny BASIC failed to run. It seems that the Motorola kit comes with no memory (except for the Mikbug private RAM), and very little else. When the user adds a 4K memory board care should be taken that all of the address and data lines are properly buffered in the expanded system, since buffers are not provided in the basic kit.

What happens is that Mikbug is able to load and display the memory with no problem, but the program will not run. This is due to the excessive capacitance in the address lines (the 6800 is spec'ed at 130pf, which is good for about 8-10 MOS devices; a 4K static RAM board alone has 32 MOS devices on some of the address lines). This causes the access lines to be slowed considerably. Mikbug does all its memory access using the Indexed addressing mode, which leaves the address stable for two full memory cycles (2 μs min) before attempting a read or write, thus permitting an actual access time of over 2.5 μs; program permitting an actual access time of over 2.5 μs; program execution on the other hand is not so forgiving, and the memory must respond in 575ns. The unbuffered system can't hack the speed. That this is indeed the problem may be verified by stretching 02 to 2 or 3 μs.

I have no record of Mr. Mikel's having attempted to communicate his problem to me, and I do know of over 100 properly buffered MEK systems on which Tiny runs fine.

Tom Pittman
Itty Bitty Computers
PO Box 23189
San Jose, CA 95123

cc: Roger Mikel
Computer Hobbyist Group-NT

A SPECIAL PURPOSE EDITOR FOR MANUSCRIPT PREPARATION?

Dear Jim, Nov. 6, 1976

About reinventing the wheel . . . am I going to have to write my own program for word processing — in the sense of manuscript preparation? Text editors are fine for programmers but they aren't of much help for authors. What is available for an 8080 or Z-80 in the public domain? F.J. Greeb's "Classy 8080 Text Editor," *DDJ* No. 6, looks like a good step in the right direction. Everything is done on the video screen except the final hardcopy output. But a manuscript processor needs to be sentence and paragraph oriented, not line oriented, and needs to have the capability of juggling stuff among tape units or floppy disk files. (I always seem to be moving paragraphs from the end of the text to the beginning or some other spot several pages away.) Then there are nice things like automatic page numbering, single or double spacing from the same source file, and the ability to not mess up special formats such as tables or lists while at the same time properly adjusting lines and paragraphs as words or sentences are added or deleted. I would be pleased to hear from anyone with interests along these lines.

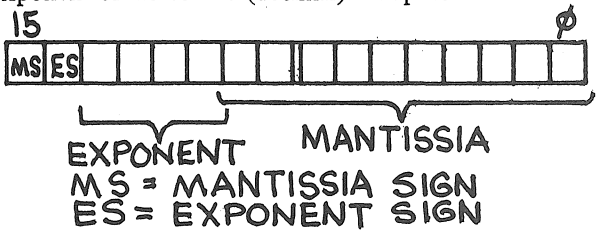
Yours truly,

Dr. Charles F. Douds
381 Poplar St.
Winnetka, IL 60093

A 16-BIT FLOATING POINT PROPOSAL

In past weeks, I have talked to several members of the CACHE about "tiny languages." I keep hearing, "I'd use it if it only had floating point." Having written three languages myself, I can understand this. Nobody seems to realize that 32 bits are a lot more than twice as hard to work with as 16.

As a compromise I propose 16 bit floating point. The format I have worked out gives 3 significant digits with an exponent of -15 to +15 (decimal). Proposed format:



I don't have the time or ambition to write this now, but I would be happy to swap ideas with anyone interested.

Bob Van Valzah (312) 852-0472 (Home)
1140 Hickory Trl. (312) 971-2010 Ext. 231
Downers Grove, IL 60519 (Work)

6800 MOTOROLA FOR SIMULTANEOUS NUMBER CRUNCHING AND ANTENNA POINTING

Dear Sir, 17 Nov. 1976

Two of us here in the Northern Virginia area are interested in using a micro for some number crunching (with a peripheral calculator chip) and antenna pointing for satellite work (simultaneously). The 6800 Motorola line of chips looks like it will fill the bill due to the superior I/O configuration possible. The 8080 kinda misses the boat. So I am interested in all kinds of homebrew hardware for 6800 line compatible with SWTP line.

Sincerely,

Ellis Marshall, W4JK Rt. 1, Box 158
Front Royal, VA 22630

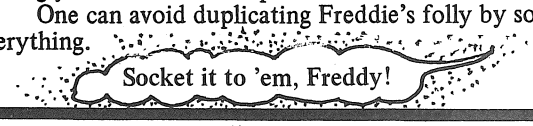
FREDDIE'S FOLLY

by Jim Day

Frugal Freddie bought a video board kit from a local computer store a couple of months ago. He saved a few bucks by not busying sockets for the ICs. "Who needs 'em?" he said. "I'll just solder everything." The board worked fine for a few weeks, then developed a hardware glitch that Freddie hasn't been able to track down. He took it back to the computer store and asked them what it would cost to fix.

"Well now," said the repairman, "If this thing had sockets, I'd probably find the trouble in a few minutes by random substitution. But with everything soldered down to the board, there's no telling how long it might take. Why, it could end up costing you more than the price of the kit!"

One can avoid duplicating Freddie's folly by socketing everything.



HAMATIC NOTE IN *BYTE*

According to a letter in the *(excellent)* November issue of *Byte*, hams who are also interested in computer phreaquery should tune to 3.865 MHz (LSB) on Thursdays at 2300 GMT "for a good time."

ERRATA FOR RANKIN'S 6502 FLOATING POINT ROUTINES

Dear Jim, Sept. 22, 1976

Subsequent to the publication of "Floating Point Routines for the 6502" (Vol. 1, No. 7) an error which I made in the LOG routine came to light which causes improper results if the argument is less than 1. The following changes will correct the error.

1. *After:* CONT JSR SWAP (1D07)
   Add: A2 00 LDX=0 LOAD X FOR HIGH BYTE OF EXPONENT
2. *After:* STA M1+1 (1D12)
   Delete: LDA = 0
   STA M1
   Add: 10 01 BPL *+3 IS EXPONENT NEGATIVE
   CA DEX YES, SET X TO $FF
   86 09 STX M1 SET UPPER BYTE OF EXPONENT
3. Changes 1 and 2 shift the code by 3 bytes so add 3 to the addresses of the constants LN10 through MHLF wherever they are referenced. For example the address of LN10 changes from 1DCD to 1DD0. Note also that the entry point for LOG10 becomes 1DBF. The routine stays within the page and hence the following routines (EXP etc.) are not affected.

Yours truly,

Roy Rankin Dept. of Mech. Eng.
Stanford University

COMPLETE 8080A FLOATING POINT PKG FOR $7.50 *AND* NEW CASSETTE DATA FORMAT STANDARD TO BE PROPOSED

Dear Editor: Sept. 21, 1976

In response to Paul Holbrook's letter in the September issue, regarding the need for a cassette data format standard, I would like to inform you that a standard with software has been developed; the Mohler standard will be published in an upcoming issue of *Interface.*

The standard allows for various types of data formats and is expandable, so new ones can be added. It is also universal enough for the format to be independent of cassette interface hardware and processor type. We hope to make the Mohler cassette format a standard in the computer hobbyist industry.

I would also like to inform readers that I have developed a single-precision floating point software package for the 8080A (6-7 digits of precision). The package includes add, subtract, multiply, divide, and utility programs to convert from ASCII BCD to binary and binary to packed BCD. It takes up about 1200 bytes and is relatively fast, e.g., 2.5 msec worst case time for multiply.

Also nearing completion is a scientific function package which includes square root, sine, cosine, exponential, natural logarithm, log base ten, arc tangent, hyperbolic sine, and hyperbolic cosine. This package is to be used with the floating point package and takes up less than 1K bytes. It also has six digits of accuracy.

The floating point package is now available for $7.50. Included are manual, paper tape, and complete annotated source listing. The scientific package will also be $7.50. Both packages may be ordered for a reduced price of $10.00. To obtain one or both, send your name, address, and the appropriate amount to:

Burt Hashizume
P.O. 172
Placentia, CA 92670

CHASE: A One or Two Player Video Game

Start it running & just try to pry your kids away from your computer

—Marvin R. Winzenread

Try to catch the bouncing dot or convert the program to a two person chase game. It requires 256 bytes of memory and a Processor Technology VDM or similar video display.

To convert to a two person chase game change addresses 0051-0056 to:

```
0051  DB  FF   IN  0FFH
0053  IF       RAR
0054  IF       RAR        Second player moves
0055  IF       RAR        using left four sense
0056  00       NOP        switches.
```

For an interesting variation in the original program change 0051 to 7B ; MOV A,E

*If your VDM is not addressed as 8C = Port Address and 8800 – 8BFF as memory, you need to change these statements.

```
0000 31 F9 00 0900    LXI 6,SP+6
0003 3E 00    1000    MVI A,0          } Blank screen
0005 D3 8C    1010    OUT 8CH *
0007 21 00 88 1020 AG LXI H,8800H*
000A 36 20    1030 ST MVI M,' '
000C 23       1040    INX H
000D 7C       1050    MOV A,H
000E FE 8C    1060    CPI 8CH
0010 C2 0A 00 1070    JNZ ST
0013 21 FD 00 1100    LXI H,F0         } Store fonts for cursors
0016 36 07    1110    MVI M,7
0018 23       1120    INX H
0019 36 0E    1130    MVI M,0EH
001B CD C1 00 1140    CALL INIT        } Place cursors on screen
001E 3A FD 00 1150    LDA F0
0021 77       1160    MOV M,A
0022 EB       1170    XCHG
0023 CD C1 00 1180    CALL INIT
0026 3A FE 00 1190    LDA F0+1
0029 77       1200    MOV M,A
002A 78       2020 N4 MOV A,B          } WAIT
002B E6 03    2023    ANI 3
002D CA 36 00 2025    JZ N5
0030 01 FF 1F 2027    LXI B,1FFFH
0033 C3 39 00 2030    JMP N3
0036 01 FF 0F 2033 N5 LXI B,0FFFH
0039 0B       2040 N3 DCX B
003A 78       2050    MOV A,B
003B FE 07    2055    CPI A,0
003D C2 39 00 2060    JNZ N3
0040 DB FF    3000    IN 0FFH          } First—player moves using the right
0042 CD 6A 00 3010    CALL MV            four sense switches
0045 3A FE 00 3020    LDA F0+1
0048 BE       3030    CMP M
0049 CA 8D 00 3040    JZ HLT
004C 3A FD 00 3050    LDA F0
004F 77       3060    MOV M,A
0050 EB       3070    XCHG
0051 7D       3080    MOV A,L          } Next—computer moves
0052 32 FC 00 3081    STA SH+3
0055 CD CE 00 3082    CALL RND
0058 CD 6A 00 3085    CALL MV
005B 3A FD 00 3090    LDA F0
005E BE       3100    CMP M
005F CA 8D 00 3110    JZ HLT
0062 3A FE 00 3120    LDA F0+1
0065 77       3130    MOV M,A
0066 EB       3140    XCHG
0067 C3 2A 00 5000    JMP N4
                                       —SUBROUTINES—
006A 36 20    6000 MV MVI M,20H
006C 32 FF 00 6010    STA S            } Move cursor
006F E6 01    6020    ANI 1
0071 C4 A9 00 6030    CNZ UP
0074 3A FF 00 6040    LDA S
0077 E6 02    6050    ANI 2
0079 C4 B6 00 6060    CNZ DN
007C 3A FF 00 6070    LDA S
007F E6 04    6080    ANI 4
0081 C4 99 00 6090    CNZ RT
0084 3A FF 00 6100    LDA S
0087 E6 08    6110    ANI 8
0089 C4 A1 00 6120    CNZ LF
008C C9       6130    RET
008D 36 04    7100 HLT MVI M,4         } If caught display '⍼' and wait for
008F DB FF    7110 A6 IN 0FFH            restart. To restart program push
0091 FE 00    7120    CPI 0              all sense switches down.
0093 C2 8F 00 7130    JNZ A6
0096 C3 07 00 7140    JMP AG
0099 23       8100 RT INX H            } Right
009A 7C       8110    MOV A,H
009B FE 8C    8120    CPI 8CH *
009D C0       8130    RNZ
009E 26 88    8140    MVI H,88H*
00A0 C9       8190    RET
00A1 2B       8200 LF DCX H            } Left
00A2 7C       8210    MOV A,H
00A3 FE 87    8220    CPI 87H *
00A5 C0       8230    RNZ
00A6 26 8B    8240    MVI H,8BH*
00A8 C9       8290    RET
00A9 01 C0 FF 8300 UP LXI B,0FFC0H     } Up
00AC 09       8310    DAD B
00AD 7C       8320    MOV A,H
00AE E6 FB    8330    ANI 0FBH *
00B0 F6 08    8340    ORI 08H *
00B2 67       8350    MOV H,A
00B3 06 03    8355    MVI B,3
00B5 C9       8390    RET
```

WHIPPLE & ARNOLD DEVELOP A SUPER DUPER BASIC INTERPRETER ($25)

Binary Systems Corp. has intrduced a new interpreter for 8080-based microcomputers. Called BASIC ETC, the new interpreter was co-developed by John Arnold and Dick Whipple of Tyler, TX, authors of the first implementation of Tiny BASIC. It includes floating point (6 to 72 digits) and variable-length integers.

"Our goal was to develop a variant of BASIC designed specificically for the hobbyist and small business user, keeping in mind that the most important priorities – from the user's standpoint – were ease of program development and straightforward, one-step program execution."

"We feel we've accomplished that goal, and with a memory efficient program, too." said Arnold.

BASIC ETC uses the lower 8K of memory plus at least 1K of RAM for scratchpad. Since BASIC ETC is for games and business applications, the less frequently used scientific functions of Dartmough BASIC are not available.

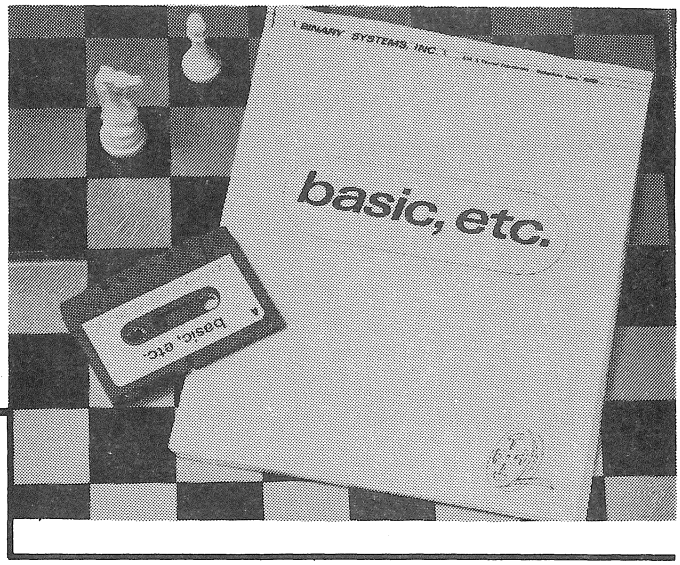
According to Arnold, BASIC ETC is readily software adapted to the individual's system, and "the best answer today for the 8080-based microcomputer owner shopping for an easy to use high level language."

The BASIC ETC kit, which includes the program – on either audio cassette tape or paper tape – and a 32-page, detailed user's manual, sells for $25.00. The manual sells for $6.00 separately.

Kits may be ordered from the Micro Store, 634 S. Central Expressway, Richardson TX 75080. The Micro Store is the retail affiliate of Richardson-based Binary Systems, Inc. Orders should include a check or money order for the price of the item. For cassette tape, the purchaser must indicate his choice of either the Kansas City or Suding/Digital Group recording technique.

Features of BASIC ETC are listed below:

* Immediate delivery
* Readily software adapted to user's system
* Resides in only 8K of memory
* Supplied on either cassette tape (Kansas City or Suding/Digital Group format), or on paper tape.
* Thorough explanatory manual.
* Full string capability – up to 255 characters string variable
* N-dimensional arrays
* Variable precision arithmetic
* Easily handles assembly language routines
* Direct memory and I/O addressing
* 27 error codes
* Both character and line erasure editing
* Subroutine nesting permitted
* 31 commands and statements
* 8 functions plus user defined functions
* Null control: 0 to 25 seconds
* Formatted output statements



Down

For random initialization of cursors

Random number generator from PCC

```
00B6 01 40 00   8400 DN LXI B,40H
00B9 09         8410 DAD B
00BA 7C         8420 MOV A,H
00BB E6 FB      8430 ANI 0FBH  *
00BD 67         8440 MOV H,A
00BE 06 03      8445 MVI B,3
00C0 C9         8490 RET
00C1 CD CE 00   8500 INIT CALL RND
00C4 E6 8B      8510 ANI 8BH   **
00C6 F6 88      8515 ORI 88H
00C8 67         8520 MOV H,A
00C9 CD CE 00   8530 CALL RND
00CC 6F         8540 MOV L,A
00CD C9         8550 RET
00CE E5         8600 RND PUSH H
00CF 21 FC 00   8610 LXI H,SH+3
00D2 06 08      8620 MVI B,8
00D4 7E         8630 MOV A,M
00D5 07         8640 RTOP RLC
00D6 07         8641 RLC
00D7 07         8642 RLC
00D8 AE         8650 XRA M
00D9 17         8660 RAL
00DA 17         8661 RAL
00DB 2D         8670 DCR L
00DC 2D         8671 DCR L
00DD 2D         8672 DCR L
00DE 7E         8680 MOV A,M
00DF 17         8690 RAL
00E0 77         8695 MOV M,A
00E1 2C         8700 INR L
00E2 7E         8710 MOV A,M
00E3 17         8720 RAL
00E4 77         8730 MOV M,A
00E5 2C         8740 INR L
00E6 7E         8750 MOV A,M
00E7 17         8760 RAL
00E8 77         8770 MOV M,A
00E9 2C         8780 INR L
00EA 7E         8790 MOV A,M
00EB 17         8800 RAL
00EC 77         8810 MOV M,A
00ED 05         8820 DCR B
00EE C2 D5 00   8830 JNZ RTOP
00F1 E1         8840 POP H
00F2 C9         8850 RET
00F3            8990 SP DS 6
00F9            9000 SH DS 4
00FD            9010 FO DS 2
00FF            9020 S DS 1
```

LIFE'S LIKE THAT

LIFE ON AN 8080 WITH A VDM

The game of life seems to be a natural for the VDM. So much has been written about it.

Here is a short version that requires toggling only 116 bytes. An earlier version (*PCC*, Vol. 4 No. 2) required 218 bytes. This program does however RAM equal to the VDM memory to store the next generation. If you are really strapped for memory, use half of the VDM for each generation.

1) Before loading the program, first initialize the screen. On Processor Technology's VDM this is done by sending a zero out to the VDM output port.

2) Load the program and run it. This should clear the screen of random characters within 10 seconds.

3) Use the front panel to load your original population directly into memory (*=2A in hex).

4) Run the program. Every 2½ seconds a new generation will appear.

— Marvin R. Winzenread

```
0000 31 7A 00   0100 LXI SP,7A00   SET STACK POINTER
0003 11 00 1C   4100 LXI D,1C00    SET POINTER TO START OF RAM FOR COPY
0006 21 00 80   4110 LXI H,8000    POINT TO START OF VDM MEMORY
0009 D5         4160 NO PUSH D     SAVE RAM POINTER
000A 0E 00      0170 MVI C,0       INITIALIZE COUNTER FOR COUNTING NEIGHBORS
000C 2A         4080 DCX H         + & COUNT
000D CD 6F 00   4090 CALL C1       }
0010 11 C0 FF   4100 LXI D,0FFC0H  } ← & COUNT
0013 19         4110 DAD D         }
0014 CD 6F 00   4120 CALL C1       }
0017 23         4130 INX H         + & COUNT
0018 CD 6F 00   4140 CALL C1       }
001B 23         4150 INX H         + & COUNT
001C CD 6F 00   4160 CALL C1       }
001F 11 40 00   4170 LXI D,40H     } ← & COUNT
0022 19         4180 DAD D         }
0023 CD 6F 00   4190 CALL C1       }
0026 19         4200 DAD D         } ← & COUNT
0027 CD 6F 00   4210 CALL C1       }
002A 2A         4220 DCX H         + & COUNT
002B CD 6F 00   4230 CALL C1       }
002E 2A         4240 DCX H         + & COUNT
002F CD 6F 00   4250 CALL C1       }
0032 23         4251 INX H         + } RESTORE VDM POINTER
0033 11 C0 FF   4252 LXI D,0FFC0H  } ←
0036 19         4253 DAD D         } RESTORE RAM POINTER
0037 D1         4260 POP D         }
0038 CD 56 00   4270 CALL R0       } CALL REGENERATION ROUTINE
003B 23         4275 INX H         }
003C 13         4280 INX D         } INCREMENT POINTERS
003D 7A         4290 MOV A,D       }
003E E6 40      4300 ANI 40H       } IF NOT FINISHED-CONTINUE
0040 CA 09 00   4310 JZ N0         } FOR NEXT LOCATION
0043 21 00 88   5000 CP LXI H,8800H
0046 11 00 1C   5010 LXI D,1C00
0049 1A         5020 C1 LDAX D
004A 77         5030 MOV M,A
004B 23         5040 C2 INX H
004C 13         5050 INX D         } COPY NEW GENERATION TO
004D 7C         5060 MOV A,H       } SCREEN AND START OVER
004E FE 8C      5070 CPI 8CH       }
0050 C2 49 00   5080 JNZ C1        }
0053 C3 03 00   5150 JMP F1        }
0056 79         7000 R6 MOV A,C    — SUBROUTINES
0057 FE 02      7010 CPI 2         — DECIDE NEXT GENERATION
0059 CA 6C 00   7020 JZ R3         } FOR THIS LOCATION
005C D2 63 00   7030 JNC R1        }
005F 3E 20      7040 R2 MVI A,20H  } IF LESS THAN 2 NEIGHBORS,
0061 12         7045 STAX D        } EMIT A SPACE
0062 C9         7050 RET           }
0063 FE 03      7060 R1 CPI 3      } IF MORE THAN 3 NEIGHBORS,
0065 C2 5F 00   7070 JNZ R2        } EMIT A SPACE
0068 3E 2A      7080 MVI A,'*'     } IF THREE NEIGHBORS EMIT A "*".
006A 12         7090 STAX D        }
006B C9         7100 RET           }
006C 7E         7110 R3 MOV A,M    } IF 2 NEIGHBORS,
006D 12         7120 STAX D        } COPY OLD CONTENTS
006E C9         7130 RET           }
006F 7E         8000 C1 MOV A,M    — COUNTING ROUTINE
0070 FE 2A      8010 CPI '*'       } INCREMENT COUNTER IF
0072 C0         8020 RNZ           } LOCATION IS OCCUPIED BY "*".
0073 0C         8030 INR C         }
0074 C9         8040 RET           }
0075            9100 N6M EQU 1C00H
0075            9110 VDM EQU 8800H
0075            9120 SP DS 6
```

KIM-1 OWNERS: PLEASE, THINK SMALL

Now that you have your KIM-1 attached to the power supply and have successfully added 2 + 3 and gotten 5, would you like something else to do? Would you like to use the KIM-1 as:

- a TIMER accurate to a millisecond
- a CLOCK displaying hours, minutes and seconds
- an ADDING MACHINE with six digit add/subtract for the old checkbook
- a DECIMAL-HEX/HEX-DECIMAL Converter
- a DRUNK TEST
- a simple GUESS-THE-NUMBER game for the kiddies
- the MASTERMIND game for you
- the SHOOTING STARS puzzle
- a series of REACTION TIME tests
- a MOVING MESSAGES DISPLAY with Alphabetic Characters
- plus other demos, tests and games??

Would you appreciate having all of these capabilities in an integrated software package that includes a "high level language" which will let you create your own programs???

MicroCosmos announces PLEASE, a package which contains all of the above features and runs on the basic KIM-1 — no additional memory, TTY, or peripherals required. PLEASE is distributed as a CASSETTE TAPE, plus complete SOURCE LISTINGS, full OPERATING INSTRUCTIONS, and instructions for writing your own programs in PLEASE. The total cost: $10.00. MicroCosmos, 210 Daniel Webster Highway, S., S. Nashua, NH 03060, (617) 256-3649.

4K AND 8K BASIC FROM SWTPC FOR UNDER $5—$10

Southwest Technical Products Corporation has just released its 4K and 8K BASIC software. Both feature fixed and floating point math with a full 1.0E-99 to 9.9999999999E+99 number range. In addition to the line number mode a direct (no line number) mode of execution is provided on most statements to create a calculator like mode of entry for short programs. Provisions have been made in both packages for saving and loading BASIC programs to and from either cassette or paper tape. A USER function is even provided for jumping to machine language subroutines.

Both packages have been written for the SWTPC 6800 Computer System. The 4K BASIC © requires a minimum of 6K of memory with 8K recommended, while the 8K BASIC © requires a minimun of 8K of memory with 12K recommended. The 4K BASIC © tape *and* manual sell for $4.95 on "Kansas City" cassette tape and $10.00 on paper tape. The 8K © tape *and* manual sell for $9.95 on "Kansas City" cassette tape and $20.00 for paper tape. All prices are postpaid in the U.S. SWTPC, 219 W. Rhapsody, San Antonio, TX 78216, (512) 344-9778

SWTPC Has copyrighted 4K and 8K BASIC. Version 1.0 program material and manual *may be copied for personal use only.* No duplication or modification for commercial use of any kind is authorized.

COMMANDS

LIST
RUN
NEW
SAVE
LOAD
PATCH

STATEMENTS

REM	END		
DIM	GOTO*	STOP	
DATA	ON . . . GOTO*	GOSUB*	
READ	ON . . . GOSUB*	PATCH*	
RESTORE	IF . . . THEN*	RETURN	
LET*	INPUT	†DES	
FOR	PRINT*	†PEEK	
	NEXT	†POKE	

FUNCTIONS

ABS	†VAL	†SIN
INT	†EXTS	†COS
RND	†LENS	†TAN
SGN	†LEFTS	†EXP
CHR	†MIDS	†LOG
USER	†RIGHTS	†SQR
TAB		

**Direct Mode statements*
† 8K Version only

MATH OPERATORS

- (unary) Negate
* Multiplication
/ Division
+ Addition
— Subtraction
†↑ Exponent

RELATIONAL OPERATORS

=	Equal
<>	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
>=	Greater Than or Equal

Tiny BASIC Game Contest

OPPORTUNITY TO WIN A MICROCOMPUTER ASSOCIATES VIDEO TERMINAL, ETC.

1st Prize: VT-200 terminal with resident TINY BASIC and JOLT assembler
2nd Prize: VT-100 terminal
3rd Prize: JOLT 4K system kit
4th - 10th Prizes: JOLT CPU kits

CONTEST RULES:

1. All entries must be postmarked by April 1, 1977.
2. All entries must be submitted as follows:
   a. JOLT TINY BASIC source program as paper tape with CR, LF and four (4) rubout characters terminating each source statement.
   b. Running instructions, game description and at least one example of game play-all in typewritten form on 8½" x 11" white bond suitable for printing.
3. All entries must run on an MAI VT-200 equipped with 4,096 bytes of RAM storage, OR on a JOLT 4K system equipped with TINY BASIC.
4. All entries must run correctly and be sufficiently well documented to enable a non-technical person to enter, run and play the game as directed by the running instructions. Entries which for any reason do not run or are not sufficiently well documented to enable easy entry and play will be DISQUALIFIED.
5. All decisions by MAI with respect to acceptance, disqualification, and winners will be final.
6. MAI employees and their families are not eligible to enter.
7. This contest void where prohibited by law.
8. All entries become the property of MAI and will not be returned.
9. Contest winners will be notified by registered mail no later than 60 days from contest closing date of April 1, 1977. Contest Winners will also be published in the *Microcomputer Digest* and the *JOLT Users Newsletter.* Contest Winners may also be obtained directly by sending a stamped self addressed envelope to MAI no earlier than May 1, 1977 and no later than July 1, 1977.
10. The JOLT TINY BASIC language summary is available at participating computer stores. The language summary may also be obtained by sending $1.00 cash, check, or money order for postage and handling to MAI TINY BASIC CONTEST, P.O. Box 1167, Cupertino, CA 95014. A paper tape form of JOLT TINY BASIC complete with documentation is available by sending $5.00 cash, check or money order to ITTY BITTY Computers, P.O. Box 23189, San Jose, CA 95123.

GOOD POINTERS ON 6800 SYSTEMS SOFTWARE

Dear Dr. Dobb's, Sept. 22, 1976

Your readers who are interested in the article by Tom Pittman on the 6800 Resident Assembler and Editor might like to know that true annotated assembly listings of the I/O routines are available in the 6800 users group library.

Program No. 10 is a listing of the I/O routines used with EXBUG. While this listing does not describe the routines in EXBUG itself, the comments do provide an insight into the operation of the flags.

Of more interest is Program No. 11 which is the MIKBUG version of the I/O routines. When this is combined with the listing of MIKBUG in Engineering Note 100 on the MCM 6830L7 ROM one will have a listing of a complete I/O system. This can be used as a model to develop suitable I/O routines to interface the Assembler and Editor with any system.

The price Motorola charges for the Assembler and Editor is a little high for home use though.

Sincerely,

John P. Byrns
1953 Governors Ln.
Hoffman Estates, IL 60195

A $5 WUMPUS

Hi —

I have written a machine language version of "Wumpus" by Greg Yob. It's a great game. The 8080 program is under 3K and is self-contained. It requires no user PROM subroutines, etc. Anyway, if anyone wants a listing, just send your name, address and $5.00 to:

Ron Santore
1957 Huasna Dr.
San Luis Obispo, CA 94301

MUMPS IS SPREADING

Oct. 18, 1976

The MUMPS computer language is used for medical and business applications. The number of institutions that use MUMPS is growing by about 80% per year. A concise pocket guide to MUMPS has been written to facilitate use of this text-handling and data management language. The guide includes descriptions of all the commands, operators, functions, and all other capabilities of Standard MUMPS, and gives many examples of their use. The Standard was developed from a dozen MUMPS dialects, under the sponsorship of the National Bureau of Standards and the Department of Health, Education and Welfare. Single copies of the guide are available at no charge from Dr. Joan Zimmerman, MUMPS Users' Group, 700 S. Euclid Ave., St. Louis, MO 63110.

PROGRAM REPOSITORY & TAPE DUPLICATION FACILITY

A PUBLIC DOMAIN ALTERNATIVE TO MANUFACTURERS' USER GROUPS

ERRATA FOR PREVIOUS CCC INFORMATION:

The CCC Program Repository currently furnishes programs on *roll* paper tape; *not* on *fan-fold*, as was previously announced.

The Community Computer Center (CCC) will act as a repository for program tapes; both source tapes and binary tapes. Everyone wishing to contribute programs to the public domain may do so by forwarding appropriate paper tapes to CCC. In particular, if you are hesitant about submitting a program for publication in *Dr. Dobb's Journal* because you don't want to hassle with its distribution, you are encouraged to forward the tapes to CCC and the documentation to the *Journal* for publication.

The CCC will thus serve as a desirable alternative and supplement to the User Groups that are controlled and operated by many of the processor manufacturers, some of whom charge up to $100 for "membership" and access to the programs that their *customers* developed and offered to the User Group, without compensation.

There is *no* membership fee for access to the tapes from the Community Computer Center. Instead, one pays only for the duplication and mailing costs:

Duplication charge: $1/ounce or fraction thereof, for tapes (weighed after punching on roll paper tape)
(Add 6% tax for orders mailed to a California address)
Postage and handling: $0.50 on orders of $5 and less
$1 on orders exceeding $5

Payment must accompany all orders. Orders will be mailed First Class, within 3 days of receipt.

Lists of available tapes will be published, periodically, in *Dr. Dobb's Journal*, as well as being available from CCC:

Community Computer Center
1919 Menalto Avenue
Menlo Park, CA 94025
(415) 326-4444

The following source tapes are currently available. They are programs written for the version of BASIC that is implemented for the HP 2000F minicomputers, and are discussed in *What To Do After You Hit Return* (available from the PCC Bookstore, $6.95).

Tiny BASIC for Altairs & IMSAIs:

Palo Alto Tiny BASIC	**2**
Star Trek in Palo Alto Tiny BASIC	**2**
Palo Alto Tiny BASIC for HP2100 XASYM	**2**

Numbers Guessing Games	$12
Number	2
Abase	3
Trap	2
Stars	2
Clocks	3
Bagels	2
Quadgt	3
Button	2
Word Games	$10
Letter	2
Abagel	3
Hangmn	3
Madlib	6
Word	2
"Nimlike" Games	$11
23Mtch	2
Batnum	3
Nim	4
Chomp	3
Zot	5
Hide-n-Seek in 2D	$ 4
Hurkle	2
Mugwmp	2
Snark	2
Pattern Games	$11
Dangle	2
Sunsgn	3
Biosin	3
Mandal	3
Life	3
Amaze	3
Board Games	$11
Qubic5	5
Gomoku	4
Teaser	3
Rover	5
Welcome to the Caves	$ 9
Caves1	5
Wumpus	4
Caves2	5
Business & Social Science	$22
Hamrbi	3
King	5
Civil2	7
Market	5
Stock	5
Policy	4
Polut	4
Science Fiction Games	$12
Trader	10
Sttr1	9
Last Chapter	$10
Crash	4
Lunar	3
Revers	2
Zeros	3
Taxman	3
The following games are in Dartmouth BASIC	
Motie	5
Rescue	5
Pounce1	
Dodgem	**3**
Sinners	**2**

Kingdom for TSS/8 BASIC:

English Version	**$ 2**
Spanish Version	**2**

SAN FRANCISCO'S SETH IS BECOMING THE BOOTSTRAP COMPUTER STORE

A computer mob known as SETH, 4001 - 24th St., San Francisco, CA 94114, is working on opening a storefront computer operation that will include walk-in, play-a-computer-game facilities. They have miscellaneous peripheral gear and would like to trade some of it for other goodies. They will also sell gear on a consignment basis. They can be contacted at the above address or at 3981 - 24th St. By phone, call (415) 282-8000 or 282-3550 (11 a.m. - 7 p.m.), and ask for Bob, George or Don.

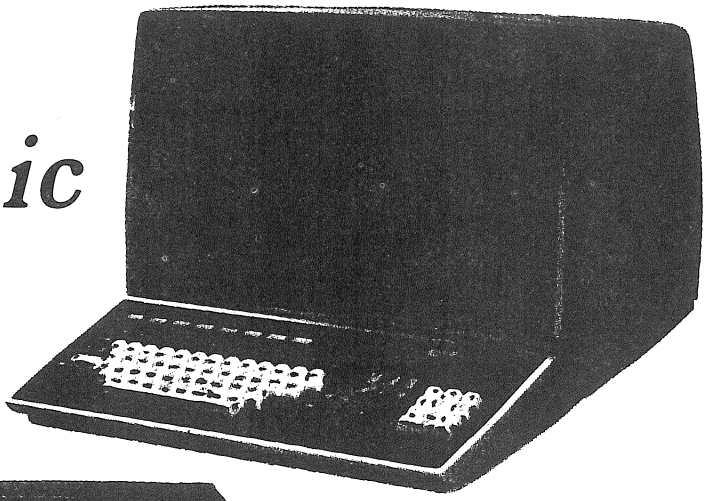
BUSINESS SOFTWARE . . . FOR $3,000

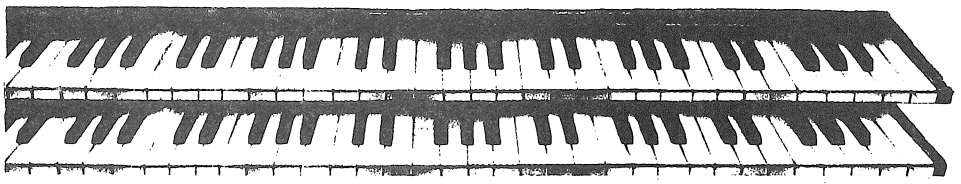
Aircom, Inc. (Rt. 16B, Union, NH 03887, 603/473-2323) has three software or business packages for business users. All are assembler coded for a Computer Automation Alpha LSI-2 and are teletype-oriented, both for I/O and for "record storage" (i.e., on paper tape!).

Their general ledger accounting program system is $3,000 for the software, alone, ot $9,950 for the software and a computer with 16K words. Their payroll package — with 38 character variables — requires 6K and is available for $3K for the software, or $8250 including an 8K machine. They also have a line-oriented forms package for $7,950 with an 8K machine or $2,700 for the software, alone.

They have no documentation that they could provide for our examination, and plan on customer training at their site in New Hampshire.

Computer Music Journal





The *Computer Music Journal* will be devoted to the development of computer systems which are capable of producing high quality music. The following topics will be covered:

* production of natural sounding timbre or quality of tone by Fourier like synthesis (with up to 128 ultra low distortion sine waves from one digital oscillator) , FM synthesis, and new methods
* design of real time playing instruments
* real time controllers such as organ like keyboards, joysticks, pressure sensitive pads, and new designs
* circuit design of microprocessor or minicomputer controlled digital oscillators (any waveshape)
* high speed multiplication (16 bit X 16 bit → 16 bit product in less than 200 ns)
* review of hardware components
* composition of music using a computer
* music theory which would be more easily realized with a computer than with traditional instruments
* homebrew digital music instruments
* choral effects
* digital filtering
* envelope generation of any shape
* digital reverberation and movement of spacial location with Doppler shifting
* high resolution, high speed digital to analog converters
* analysis of acoustic instruments
* psychoacoustics
* reviews of books about computer music, acoustics of musical instruments, psychoacoustics, music theory, computer design, and electronics.

The first issue of the journal will be about 50 pages in length. If enough people subscribe to pay for printing a larger journal, the journal will increase in size. A one year subscription will cost $14 and be published by PCC non profit. The journal will be published every other month. The first issue will be mailed out during January, 1977.

If interested please mail to: PCC, Box E, Menlo Park, Ca. 94025

.....

☐ Enclosed is $14 for a one year subscription to the *Computer Music Journal*

Name _____

Address _____

City _____ State _____ Zip _____

Your interests ?

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

SECOND CLASS MAIL
MAGAZINE

TIME VALUE: PLEASE
DO NOT DELAY