September/October 1981 Volume1, Number 5 $2.50 (USA) £1.20 (UK)

The magazine for Sinclair users

SYNC

Reviews and Resources:
- 16K RAM Schematic
- Hints and Tips for the ZX81
- Q S Soundboard

Games:
- Cannonade
- Robot Composer
- Defuse
- Hangman
- Motorcycle Race

Graphics and Math:
- GRA+PICS
- Prime Numbers
- Great Circle Route
- ZX80/1 As Fortune Cookie

Programming:
- PEEK and POKE
- READ on the ZX80
- Key and Token Use

The magazine for Sinclair users

SYNC

September/October 1981 Volume 1, Number 5

DEPARTMENTS

GRAPHICS AND MATH

PROGRAMMING

GAMES AND PROGRAMS

REVIEWS AND RESOURCES

Staff

Publisher/Editor-in-Chief	**David H. Ahl**
Editorial Director	**George Blank**
Managing Editor	**Paul Grosjean**
Associate Editor	**David Lubar**
Secretary	**Elizabeth Magin**
Production Manager	**Laura MacKenzie**
Art Director	**Susan Gendzwil**
Assistant Art Director	**Diana Negri**
Typesetters	**Jean Ann Vokoun** **Maureen Welsh**
Financial Coordinator	**William L. Baumann**
Bookkeeper	**Patricia Kennelly**
Customer Service	**Ralph Lovery**
Order Processing	**Ruth Coles**
Circulation	**Frances Miskovich** **Dorothy Staples** **Carol Vita**

MEMBER MITA

Index to Advertisers

Volume 1, Number 5

SYNC is published bi-monthly for $10.00 per year by Creative Computing, 39 E. Hanover Ave., Morris Plains, NJ 07950. Second class postage paid at Morris Plains, New Jersey 07950, and additional entry offices.

Postmaster: Send address changes to SYNC, P.O. Box 789-M, Morristown, NJ 07960.

Subscriptions in USA: 6 issues $10; 12 issues $18; 18 issues $24. UK and foreign airmail subscriptions: 6 issues £10; 12 issues £18; 18 issues £25. Call (800) 631-8112 toll-free (in NJ, 201-540-0445) to begin your subscription.

The Cover

The cover shows the Sinclair ZX81 computer. Although available in England since March 1981, it was formally introduced to the U.S. market on October 7, 1981.

letters

REM on the 4K ROM

Dear Editor:

I have been visiting my brother this summer and learning a little about programming the ZX80. Recently I discovered something interesting:

```
10 REM
20 PRINT "THIS LINE WILL NOT PRINT"
30 PRINT "THIS ONE WILL"
```

In this example line 20 will not print, but if line 10 is changed to

```
10 REM 1 (or any other character)
```

lines 20 and 30 will print.

We decided that the ZX80 (4K ROM) skips the next character after the remark command. If that character is a "newline," it skips the entire next line. (Note by his brother: The interpreter skips over everything up to the next NEWLINE, after skipping the character following the REM.)

I have really enjoyed your magazine and the games that are published.

David Brendel
1489 Corn Crib Court
Stone Mountain, GA 30088

Dear Editor:

I would like to warn your readers about a semi-bug in Sinclair's 4K ROM. If you use a REM, you must put something after it other than spaces. Otherwise, the computer ignores the next line of the program. I have informed Sinclair Research of the problem, and I certainly hope they correct it in later ROMs.

Geoffrey Horton
658 S. Diamond
Jacksonville, IL 62650

Ed.—When we received two letters at about the same time on the same topic, we decided to refer the question to some of our sources. Here are their replies:

David Ornstein—This feature of the 4K ROM is a powerful debugging tool. Suppose you want to check how a particular line, say line 150, is functioning in a program without going to all the trouble of deleting it and then reentering it. You simply type in line 149 REM, and RUN your program. The ZX80 will run the program and skip over line 150. You can observe the results and then easily EDIT line 149.

Joseph Sutton—The REM command can be used in a line to remove it temporarily from the program by inserting REM immediately after the line number. After you have checked the performance, you delete the REM.

James Grosjean—You can use this feature as a memory saving tool in some programs. For example, in "Motorcycle Race" in this issue, you can save six bytes by deleting lines 100-114 and substituting the following:

```
102 GO TO 102+RND(6)*2
104 PRINT "WATER AND MUD"
105 REM
106 PRINT "DEEP HOLE"
107 REM
108 PRINT "SHARP TURN"
109 REM
110 PRINT "BUMPY TRACK"
111 REM
112 PRINT "FALLEN RIDER"
113 REM
114 PRINT "LOOSE GRAVEL"
```

"Mini-Billboard" for 8K ROM

Dear Editor:

When I received my July/August issue of *SYNC* magazine, one of the first programs I looked at was "Mini-Billboard." Since I now have the 8K Basic installed and I do not want to go back, I looked it over to see whether it would be easy to convert.

The only serious problem was finding where the character generator was located in the 8K ROM. A few weeks before, I had obtained a listing of the 8K ROM, so I looked for the patterns necessary for the character generator. Then all I needed was the address. I picked what looked like a unique pattern just before the beginning of the character generator and wrote the program:

```
10 FOR A=3 TO 9E3
20 IF PEEK A=0 AND PEEK(A-1)=255
AND PEEK(A-2)=255 AND PEEK(A-3)=
201 THEN PRINT A
30 NEXT A
```

(If you are interested in using this, it takes almost 5 minutes to run). Now armed with the address (7680) I re-wrote lines 20, 70, and 110 and removed line 21.

If you change line 10 of the memory search program to 17E3 in place of 9E3, you will find the pattern occurs twice, at 7680 and again at 7680 plus 8K (or 8192 2^{13} = 15872). I would be interested in finding out why a particular byte in the ZX80 ROM can be addressed at more than one place.

The "Mini-Billboard" program revised for the 8K ROM is as follows:

```
 5 DIM A(8)
10 INPUT A$
15 FOR I=1 TO 8
20 LET A(I)=(CODE(A$(I)*8)
   +7680
23 NEXT I
25 LET F=1
27 LET L=4
30 FOR X=Ø TO 7
35 FOR I=F TO L
40 LET C=PEEK(A(I)+X)
50 FOR Y=Ø TO &
60 LET E=2**(7-Y)
70 IF C =E THEN GO TO 100
80 PRINT "#";
90 GO TO 120
100 LET C=C-E
110 PRINT "■ "; (shift 9; hit
                 space)
120 NEXT Y
130 NEXT I
140 NEXT X
150 LET F=F+4
160 LET L=L+4
170 IF L=8 THEN GO TO 30
```

My thanks to *SYNC* for a great magazine and to Dennis Duke for a great program.

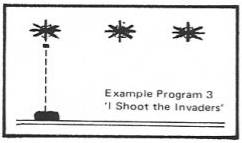
Joseph R. Sutton
170 S. Hillside Ave.
Succasunna, NJ 07876

ZX80

JRS SOFTWARE

19 WAYSIDE AVENUE, WORTHING, SUSSEX, BN13 3JU
TELEPHONE WORTHING 65691 (Evenings and Weekends only)

ZX81



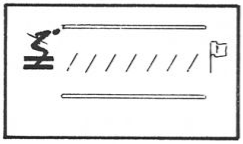
ZX80 - PROGRAMMABLE MOVING DISPLAY

(4K-ROM only)

Yes! This really is a **genuine** moving display, **not** another pause routine. If you want moving, flicker free displays [*and who doesn't*] then this is the program for you. The secret lies in the ZX80's ability to keep the display on your screen without the need to use all of the time available to it. Normally the ZX80 would be doing nothing during this spare time but the programmable moving display cleverly interupts to process your own instructions written in the simple but highly effective JRS numeric code. Great care has been taken so that the processing of your codes can always be interupted to return to the display routine at the precise microsecond that is required to ensure that your T.V. picture remains completely **rock-steady.**

Normally a true moving display on a ZX80 would take weeks to write and you would need to be an expert at machine-code programming. Now, at last, this program offers you the ability to write your own true moving displays in under an hour with no maching-code experience required whatsoever.

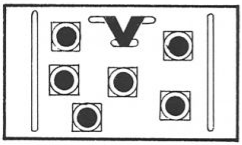
Cassette with 1k, 2k versions and 3 example programs plus FULL documentation **£4.95**



ZX81 - SLALOM (16K RAM PACK REQD.)

Slalom events always draw great crowds to the ski resorts and the T.V. cameras are never far behind. **Now** the skier on your T.V. screen is directly under your control and his success in negotiating the slalom posts and achieving a fast time relies entirely on your skill with the ZX81 keys.

Cassette and instructions **£2.95**



ZX81 - BLACK HOLES (16K RAM PACK REQD.)

Your starship is in an unknown galaxy consisting entirely of black holes which continually threaten to swallow you. Your skill at the controls and your ability to look and think many moves ahead is the only thing that stands between you and destruction. How long can you survive!

Cassette and instructions **£2.95**

SPECIAL OFFER	SLALOM and BLACK HOLES on one cassette for only £4.50
OVERSEAS CUSTOMERS PLEASE NOTE	Payment must be made in Sterling by International Money Order (available at your bank) Please add 50 pence to cover overseas postage.

"Bar Charts" and Rubber Cement

Dear Editor:

In my program "Setting Up Bar Charts" (*SYNC* 1:4), additional data can be entered in the immediate mode, and the REM on line 320 can be eliminated, since J will always point to the first element of B which contains a zero. Example: LET B(J) = 1104. However, the program must be run (using GO TO 1) to update the value of J each time a value is added to vector B.

For those having a problem with putting the 8K keyboard template over the old keyboard, rubber cement, a temporary paper adhesive, seems to do the job. The cement remains flexible, and the template can easily be removed later if desired. The best procedure seems to be to lay a thin coat on the keyboard, let it dry a couple of minutes, and then attach the template. Since rubber cement is highly flammable, be sure to take proper precautions.

Jon Passler
344 Cabot St.
Beverly, MA 01915

Splitting Strings

Dear Editor:

Harry Doakes' article on string handling in the July/August issue was great. However, he didn't mention one other useful technique that rises naturally from his explanations—how to split a string into sub-strings. The following program will do it.

```
 10 (enter the program shown in SYNC
1:4, p. 26)
 20 LET A$="STRING TO BE SPLIT"
 30 LET L$=""
 40 FOR F=1 TO 13
 50 LET B$=CHR$(CODE(A$))
 60 LET A$=TL$(A$)
 70 LET M$=B$
 80 LET L$=L$
 90 LET M$=M$
100 RANDOMIZE USR(16427)
110 NEXT F
120 PRINT L$
130 PRINT A$
```

This program has the effect of chipping away the leading characters from A$ and piling them up, one by one, at the end of L$. With the inputs shown above, the result will be "STRING TO BE" and "SPLIT". In practice, of course, lines 20 and 40 will be altered to fit the needs of the moment. You can also extend this technique to insert a string into the middle of an existing string.

Basil Wentworth
1413 Elliston Dr.
Bloomington, IN 47401

Short Video Cable

Dear Editor:

My son, James Willis, got a ZX80 computer for Christmas 1980, but we have a problem. The video cable with the coaxial plugs is too short for usage.

Can we obtain from the manufacturer a longer coaxial cable and where specifically do we do that?

Charles D. Willis, M.D.
2490 West Fir Ave.
Fresno, CA 93711

Ed.—Check your local electronics supply store for a standard coaxial cable with standard RCA plugs. If it does not carry one long enough for your use, the parts should be available to make one.

The 5-6 Seconds of Silence

Dear Editor:

When recording programs on cassettes, my auto record level cassette machine is fooled by the silent period prior to the issuance of the message. Due to the silence, the gain increases until it is wide open, and, when the signal is produced, it initially overloads the recording and the results are unusable. I have solved this problem by not engaging the cassette recorder until after 5-6 seconds of silence have passed.

A. Dan Klyver
29 Old Stagecoach Rd.
Weston, CN 06883

Ed.—See "Perceptions" in this issue for more on loading.

Glitchoidz Report

On Hunting Glitchoidz

A survey of Glitchoid activity reveals that they have favorite targets. In addition to spaces in PRINT statements noted in our last issue (*SYNC* 1:4, p. 3), semicolons should be carefully observed since these help arrange your display. Another place to look is in the use of parentheses and quotation marks. These must always be used in pairs. Another special place is in IF statements and the use of the "greater than" and "less than" signs. If there is a space in which one might fit, you might experiment.

Handling Character Strings (1:3, p. 6, col. 1)

```
200 LET A=USR(47)-2
```

Black Hole (1:3, p. 9)

```
52 PRINT.;I-3;"#";I-2;"#";I-1
```

Perceptions (1:4)

p. 6:
Listing 1. Comments column.
(4th line: IF HL≠DE . . .)
p. 7:
Listing 2. Hex column.
1st line: 217D40
2nd line: ED5B0C40
10th line: 48
Listing 4

```
10 REM 217D40ED5BOC4006007CBA20
087DBB2004480600C978AE472318EE
```

Machine Code Keyboard Scanning Program (1:4, p. 8)

```
60 IF B > 255 . . .
100 IF MARK < 10
```

Screen Scrolling (1:4)

p. 17, col. 3, add:
RUBOUT,RUBOUT,ENTER 50 and NEWLINE
RUBOUT,RUBOUT,ENTER 60 and NEWLINE

p. 19, The Great Glitchoid struck an almost fatal blow to Dr. Logan's *Road Game* by chopping the last line of the program.

```
480 PRINT "*#YOU CRASHED AFTER
#";C;"#MILES#*"
LIST 40 and SAVE
```

Multi-Dimensional Arrays for the ZX80 (1:4, p. 25)

```
240 IF I > 9 OR I < 0 . . .
270 IF J > 9 OR J < 0 . . .
```

Detective (1:4, p. 36)

```
200 PRINT (CHR$(A(K))):
```

Mini-Billboard (1:4, p. 44, col. 1)

```
80 PRINT "#";
```

Make the Most of Your ZX81 or 80

SYNC Magazine

SYNC, a bi-monthly magazine for users and prospective users of the Sinclair ZX80 computer has expanded its coverage to include the ZX81 as well.

Now entering its second year, **SYNC** has been providing nearly 10,000 Sinclair computer owners with information on how to make most effective use of their computers. "Resources," one of the most popular sections of the magazine, has listed over 100 second source vendors of software, peripherals and books as well as user groups.

Each issue of the magazine carries complete application programs, tips and techniques for more effective programming, hardware modifications and in-depth evaluations of software, peripherals and books.

Subscriptions to **SYNC** cost $10.00 per year (6 issues). **SYNC**, 39 E. Hanover Ave., Morris Plains, NJ 07950. (201) 540-0445.

The ZX81 Companion

The ZX81 Companion by Bob Maunder follows the same format as the popular **ZX80 Companion**. The book assists ZX81 users in four application areas: graphics, information retrieval, education and games. The book includes scores of fully documented listings of short routines as well as complete programs. For the serious user, the book also includes a disassembled listing of the ZX81 ROM Monitor.

MUSE reviewed the book and said, "Bob Maunder's **ZX80 Companion** was rightly recognized to be one of the best books published on progressive use of Sinclair's first micro. This is likely to gain a similar reputation. In its 130 pages, his attempt to show meaningful uses of the machine is brilliantly successful."

"The book has four sections with the author exploring in turn interactive graphics (gaming), information retrieval, educational computing, and the ZX81 monitor. In each case the exploration is thoughtfully written, detailed, and illustrated with meaningful programs. The educational section is the same—Bob Maunder is a teacher—and here we find sensible ideas tips, warnings and programs too."

Softbound, 5 1/2 x 8", 132 pages, $8.95.

The Gateway Guide to the ZX81 and ZX80

The Gateway Guide to the ZX81 and ZX80 by Mark Charlton contains more than 70 fully documented and explained programs for the ZX81 (or 8K ZX80). The book is a "doing book," rather than a reading one and the author encourages the reader to try things out as he goes. The book starts at a low level and assumes the ZX80 or ZX81 is the reader's first computer. However by the end, the reader will have become quite proficient.

The majority of programs in the books were written deliberately to make them easily convertible from machine to machine (ZX81, 4K ZX80 or 1K ZX80) so no matter which you have, you'll find many programs which you can run right away.

The book describes each function and statement in turn, illustrates it in a demonstration routine or program and then combines it with previously discussed material.

Softbound, 5 1/2 x 8", 172 pages, $8.95.

Computers For Kids, Sinclair Edition

Computers For Kids, by Sally Larsen is the fourth book in this highly successful series. (Previous editions have been released for TRS-80, Apple and Atari computers.) Written expressly for youngsters ages 8 to 13, the book requires no previous knowledge of algebra, variables or computers. Armed with a ZX81 and this book, a child will be able to write programs in less than an hour. A section is included for parents and teachers.

The book starts with a patient explanation of how to use the Sinclair, graduates to flow charts, and simple print programs. The twelve easy-to-read chapters go through loops, graphics and show other programming concepts, and show in a painless way how to make the computer do what you want.

Donald T. Piele, Professor of Mathematics at the University of Wisconsin-Parkside says, "**Computers For Kids** is the best material available for introducing students to their new computer. It is a perfect tool for teachers who are learning about computers and programming **with** their students. Highly recommended."

Softbound, 8 1/2 x 11", 56 pages, $3.95.

Order Today

To order any of these books, send payment plus $2.00 shipping and handling per order to Creative Computing Press at the address below. Visa, MasterCard and American Express orders should include card number and expiration date. Charge card orders may be called in toll-free to the number below.

creative computing

39 E. Hanover Avenue
Morris Plains, NJ 07950

Toll-free 800-631-8112
In NJ 201-540-0445

SYNC NOTES

Paul Grosjean

SYNC Program Listings

Readers should note the following conventions used in the program listings in this issue:

or • = Used in PRINT statements to show necessary spaces.

"A" (shift) = Used in PRINT statements to indicate graphics; in this case use the graphic on shift A.

INPUT = Used in PRINT statements to show that the keyboard key or token should be used instead of spelling out the word (Richard McDaniel's article in this issue).

In some schematics you may have noticed a designation given as 2K2. This is U.K. usage. U.S. usage is 2.2K. *SYNC* follows U.S. usage, but occasionally a U.K. schematic does not get completely converted.

8K ROM Problems

Several readers have asked for information about 8K ROM problems they had encountered. We referred the question to Sinclair, USA, and this is Nigel Searle's answer: "Some 8K ROMs have a bug. We are waiting for a new batch of correct ROMs from our supplier. This may take several weeks. In the meantime, please write your name, address, and the words, "8K ROM" on a piece of paper and send it to Sinclair Research, 50 Staniford St., Boston, MA 02114. We will then send you a new ROM as soon as they are available. We apologize for any inconvenience you may have been caused."

How do you know whether your ROM is one with the bug? The bug seems to show up only in certain arithmetic computations using large numbers. David Orstein has supplied us with a simple test: Try having your 8K ZX80 print 2^{32}. The answer shown should be 4,294,967,300 (actually 4,294,967,296). Then print 2^{32}-1. If the answer given is anything other than the first answer minus 1, you have one of the ROMs with the bug and you should send in as above. However, Nigel Searle has emphasized that you should keep your present 8K ROM until it is replaced and that it should cause no problems in normal programming.

SYNC Subscriptions

SYNC subscribers have now passed the 8000 mark. Up to this point all subscriptions have begun with the first issue, but now we have to start with the second issue.

ZX Microfair

The first show centered on the ZX80/1 computer will be held on September 26, 1981, at Central Hall, Westminster, London. Mike Johnson is organizing the show in association with the National ZX80/1 Users Club. Hardware and software suppliers will have space for their wares. "We hope," says Mike, "that the exhibition will prove an excellent market-place for the exchange of ideas and information, as well as giving people a chance to display and sell programs and peripherals."

SYNC in Microcomputer Index

SYNC is now one of the twenty microcomputer magazines included in *Microcomputer Index*, a recent entry into periodical indexing.

Since the field of microcomputers is growing so rapidly, one of our big problems is finding out what others are doing. One answer, of course, is to subscribe to every magazine that might cover the topics we are interested in. *Microcomputer Index* provides an alternative and promises to be a major resource for those people working (and/or playing) in the field. Users will be able to keep up with what is going on and to pinpoint the articles that will be of most use.

The *Index* has two sections. In the first section articles are indexed alphabetically by title under nearly 300 topic headings which are listed together in the front of the Index. The title is followed by the author's name, an abstract number, a classification by type (article, book review, column, hardware review, letter, software review), the language of the program listing, the magazine, the volume and issue numbers, and the date of issue.

In the second section the same information is given under the magazine name for each issue covered by the current index. Entries are made in the order of appearance in the magazine. In addition, an abstract of the article is given along with the list of topics under which it is indexed. These abstracts are brief, usually about 25 words, and on target (at least as far as the *SYNC* articles surveyed are concerned).

The current issue runs 96 pages and includes about 1250 articles. The *Index*, now into its second volume, is published by Microcomputer Information Services (see *SYNC* 1:4, Resources, p. 48).

SYNC NOTES: UK

Sinclair Launches the ZX Printer

Sinclair Research has introduced the long awaited ZX Printer for the ZX81 and ZX80 (8K ROM only) at the PCW show in London in early September. The printer is available only from Sinclair by mail order at £49.95 inc VAT.

ZX80/1 users will now be able to get hard copy of their listings. The printer features full alphanumerics and sophisticated high resolution graphics. The special features include COPY, which prints out exactly what is on the screen without further instructions. The operation is complete in 14 seconds at an effective cost of less than 1p. L LIST instructs the printer to produce an entire completed program, and L PRINT to print copy out on the printer and not the screen. The copy has 32 characters to the line, 9 lines to the vertical inch, and a printing speed of 50 characters per second. It is attached to the rear of the computer by a stackable connector which allows the 16K RAM pack to be used at the same time. The special aluminized paper comes in 65-foot rolls which will take care of over 250 full screens of text. The paper will be supplied in packs of five for £11.95.

Sinclair to Sell ZX81 Retail

Sinclair has signed an exclusive trial agreement to sell the ZX81 retail through over 100 W H Smith high-street stores. "Both parties view the agreement as an experiment," commented Clive Sinclair, Chairman of Sinclair Research. Under the initial five month agreement Smith's will set up new computer departments in each store with specially trained staff to demonstrate the ZX81 and give after-sales service. Sinclair products will sell at the normal prices, but the new ZX Printer and ZX81 kit will not be available for the time being.

Make the most of your Sinclair Computer . . .

Software on Cassette!

MULTIFILE — Data Storage System An amazingly versatile multi-purpose filing system for the 16K ZX81. The program is menu-driven, and number, size and headings of files are user-definable. Both string and numerical files are catered for. Files may be created, modified, replaced, and searched, and are protected by an ingenious foolproof security system. Output to the ZX printer is also provided.

The program comes on cassette, together with three quality data cassettes for file storage, and comprehensive documentation, describing a host of applications for both business and personal use. If your ZX81 is bored with playing games, then this program will give it plenty to think about! . .$29.95 ($39.95 in Canada)

ZXAS MACHINE CODE ASSEMBLER Bored with BASIC? POKEING not your scene? Learn and program in machine code the easy way with this powrful Z80 assembler, commissioned specially for the ZX81 & ZX80.

Standard Z80 nemonics are simply written into REM statements within your BASIC program. The assembly listings, together with addresses and assembled codes are displayed on the screen when assembled. The assembled code is executed with the USR function. The program uses 5K of memory and is protected from overwriting. Full documentation, including examples, is supplied with the cassette. This program is a must for all serious ZX81 & ZX80 users . . . .

Last Minute Addition: ZXDB

The perfect complement to ZXAS assembler, ZXDB is a complete combined machine code disassembler and debugging program. May be used in conjunction with ZXAS and will leave about 9K of memory for your own program. Additional features include Single Step, Block, Search, Transfer and Fill, Hex Loader, Register Display and more. Executed by single keyboard entry. The combination of ZXAS/ZXDB plus one of our books will teach you all you need to know to program in machine codes.
ZXDB . . $9.95 ($12.95 in Canada)

Exciting Book Titles!

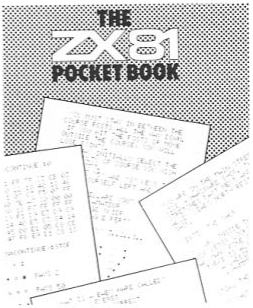
MACHINE LANGUAGE MADE SIMPLE FOR ZX80 and ZX81. A complete beginners quide to machine language programming. Go beyond BASIC and open new computer horizons! Finally find out what PEEK and POKE is all about. Machine language program enables more computing power in less space, faster running programs. The 120 pages of this book are packed with programming techniques, hints and tips; useful BASIC program to edit machine language; numerous sample routines; easy-to-use reference tables.
$19.95 ($23.95 in Canada)

UNDERSTANDING YOUR SINCLAIR ROM. A more advanced publication explaining the various ROM features.$19.95 ($23.95 in Canada)

ZX CHESS!

(for ZX81 and 8K/ZX80 both with 16K RAM)

A challenging chess programme, written in machine language, designed to operate in the ZX81 fast mode. ZX Chess allows you to select from 6 levels of play, choose either black or white, and enables castling and en passant moves. Unique "self-running" feature: you start the tape and when the chess board appears on the screen, start your game
ZX CHESS! Melbourne House. **$24.95** (29.95 in Canada)



The ZX81 Pocket Book
Written in the informative and clear style of the earlier, highly successful ZX80 Pocket Book, but with all new content. This is the ideal follow-up to the Sinclair manual, with application to both ZX81 and 8K ROM ZX80! The ZX81 Pocket Book begins with an exceptional 1K RAM programme (Pinning the Tail on the Donkey), which is followed by revealing chapters on String-Functions and Efficient Programming. Throughout there is a balance between serious computing concepts and fun programs. A particular emphasis is placed on the use of subroutines. Ohter chapters provide Hints 'n Tips, Decimal Justification, Using Machine Code, Numeric Conversion, and ZX81 Adventure. Programs for both 1K and 16K machines include: Ski Run, Ball & Bucket, Etch-a-Sketch, Digital Clock, Standard Deviation, Dice Simulation, City of Alzan (a long adventure program), plus many others. The book contains 5 appendices containing ZX80 and ZX81 conversions, ZX81 module selector listing, solutions to problems in the book, ZX81 Basic command summary, and error code summary. The emphasis throughout is on a programming style designed to conserve memory, and demonstrate practical techniques to make your programs function better. Every Sinclair owner should have a copy right alongside his manual!
The ZX81 Pocket Book, by Trevor Toms, Phipps Associates. 136 pages. Spiral bound. $11.95 ($14.95 in Canada)

NOT ONLY 30 PROGRAMS FOR THE SINCLAIR ZX81 . . . BUT ALSO . . . detailed explanations and much much more. All programs designed to fit into the 1K memory of the ZX81. Includes such favorites as Star Wars, Lunar Lander, Blackjack, Mini Adventure. Also explanations of how programs were written, hints on how to write your own exciting programs, space-saving techniques, peeks and pokes and other "complicated" functions. . . . . .
$14.95 ($16.95 in Canada)

Mail Orders to: GLADSTONE ELECTRONICS 901 Fuhrmann Blvd., Buffalo, NY 14203. (in Canada, mail to Gladstone Electronics, 1736 Avenue Rd., Toronto, Ont M5M 3Y7)

Name _____
Address _____
City _____ State _____ Zip _____
Charge to ☐ Visa ☐ MasterCard
Card No. _____
Expiry _____
☐ Check ☐ Money order (Sorry, no CODs)
Amount enclosed _____
Full replacement warranty all tapes.

Please send

Quantity	Description	Price each
	CASSETTES	
	Multifile -Data Storage System	$29.95 ($39.95 Cdn)
	ZXAS - Assembler	$9.95 ($12.95 Cdn)
	ZXDB - Disassembler/debugger	$9.95 ($12.95 Cdn)
	ZX CHESS!	$24.95 ($29.95 Cdn)
	BOOKS	
	Machine Language Made Simple for ZX80 & ZX81	$19.95 ($23.95 Cdn)
	Understanding Sinclair ROM	$19.95 ($23.95 Cdn)
	The ZX81 Pocket Book	$11.95 ($14.95 Cdn)
	Not Only 30 Programs for Sinclair ZX81	$14.95 ($16.95 Cdn)
	Shipping charge, all orders	$1.50

Currently Technical Services Manager, David Ornstein has been with Sinclair since the opening of its U.S. office. He has been involved with Sinclair's technical hotline, technical writing, and machine servicing. His primary interests are in the areas of software and hardware R & D, and system integration.

His secondary interests are reading (Frank Herbert), listening (Pink Floyd) and sharing

perceptions

David Ornstein

A ROM Munching Session

Overview

One important aspect of a computer is its I/O (Input/Output) system. The I/O functions provided by a given computer system govern the usefulness of that system. The ZX80 is slightly limited by its storage system. With a ZX80, you can SAVE a program and LOAD it back in again. Granted, your variables are SAVEd (and LOADed) with your program, so you do have some limited data storage. But, imagine a program which takes an *independent* set of data (e.g., a text file), operates on it (e.g., a text editor or compiler), and produces some arbitrary output (e.g., a machine code program). This type of program is sometimes known as a filter. The difficulty is that you have no way of getting the independent data to the program.

One solution to this problem is to have a copy of the filter (program) for each set of data. This is not a very efficient method, but it does have the advantage of simplicity. However, I believe that a user should have a choice between filter copying and independent data storage. Then he can make his own trade-off decisions, thus utilizing the better method for his application.

In this issue, I will take you on a journey through the ZX80's storage system. I will show you how to utilize the LOAD and SAVE routines in the ZX80's ROM (Read-Only Memory), and apply them wherever, and however, you want. But first I will run through, in detail, the algorithm(s) used by the ZX80's cassette system. For the greatest level of understanding you should follow the code in Listing 1 very carefully.

Listing 1.

```
Label     Assembly              Comment

                                ; Copyright (c) 1981,
                                  by Sinclair Research Ltd.
SAVE:                           ; SAVE RAM
                                ;
                                ; Address=01B6 hex
                                          0438 decimal
                                ;
                                ; Save from 4000h to (E.LINE)
                                ;
          POP DE                ; remove return address
          LD DE,4811            ; lead in
LSAV2:    LD A,7Fh              ; KB mask
          IN A, (0FEh)          ; set output to zero and
                                ; read BREAK key
LSAV0:    RRA                   ; move BREAK into CY (carry)
          JR NC,LSB             ; if BREAK pressed
          DJNZ $                ; wait 255*13+63 T-States
                                ; (1.04 ms) each time around
                                ; the LSAV2 silence loop;
                                ; i.e., 5 seconds in all
          DEC DE
          LD A,D
          OR E
          JR NZ,LSAV2           ; if DE#0
          LD HL,4000h           ; now dump RAM
LSAV27:   LD DE,8+(248,sh1.8)   ; i.e., e=8,d=248
LSAV3:    RLC (HL)              ; get next bit
          SBC A,A               ; FF if a 1, 00 if 0
          AND 5                 ; 5 if a 1,0 if 0
          ADD A,4               ;
          LD C,A                ; c:=9 if a 1, 4 if a 0
LSAV4     OUT (0FFh),A          ; tone burst
          ID B,36               ; DELAY
          DJNZ $                ; LOOP
          LD A,07Fh
          IN A, (0FEh)          ; set output to zero
                                ; and read BREAK key
          LD B,35               ; DELAY
          DJNZ $                ; LOOP
          DEC C
          JR, NZ, LSAV4
          LD B,D                ; get timing for IB gap
LSAV5:    NOP                   ; inter-bit gap
          DJNZ LSAV5
          LD D,254
          DEC E
          JR NZ. LSAV3          ; if more bits
          RRQ
          JR NC,LSB             ; if BREAK pressed
          CALL ENDBYT
          JR LSAV27
ENDBYT:
                                ; common code for LOAD and
                                ; SAVE at end of byte.
                                ; 72 T-States including call
                                ; and return if returns.
          INC HL                ; bump pointer
          EX DE, HL
          LD HL, (E-LINE)
          SCF                   ; carry = 1
          SBC HL, DE            ; HL=HL - DE - carry
          EX DE, HL
          RET NC                ; unless end of area to be dumped
          POP HL                ; discard return addr from ENDBYT
```

```
Label     Assembly          Comment

LS8:      JR NEWLIN
LOAD
                            ; LOAD RAM
                            ;
                            ; Address = 206 hex
                            ;           518 decimal
                            ; LOAD from 4000h on

          POP DE
LL 1:     LD DE, 22290
LL2:      LD 9,7Fh
          IN A, (0FEh)      ; first look for lead-in: read
          RLA               ; every 15.1 µs if 1's, 21.5 µs
          RRA               ; if 05, wait until get
                            ; zeroes for 480 ms, abort to
          JR NC, LS8        ; NEWLIN IF BREAK pressed
          RLA               ; CY = cassette data
          JR C,LL1          ; if high then no leader yet;
                            ; so go back
          DEC DE
          LD A,D
          LD A,D
          OR E
          JR NZ, LL2        ; if DE≠0
          INC (IY+E-LINE+1-   ; in case is 40h now
                  4000h)
          LD HL, 4000h
LL27:     LD E, 8
LL3:      LD A, 7F
          IN A,  (0FEh)     ; get cassette/BREAK data
          RRA               ; put it in CY (BREAK, that is)
          JR NC, LL7        ; if BREAK
          RLA
          RLA               ; CY = cassette data
          JR NC, LL3        ; read every 16 µs until a 1 is
                            ; found
          L D C, 148
LL4:      LD B, 26          ; req'd 408 µs trailer after
                            ; tone burst
LL5:      DEC C             ; count 15.4 µs if a 1, 15.2 if a 0
          IN A, (0FEh)      ; N.B. C is decremented at least
                            ; 26 times, i.e., to 122 which
                            ; has bit 7=0
          RLA
          BIT 7,C           ; test bit 7
          LD A,C
          JR C, LL4
          DJNZ LL5          ; until end of trailer then
                            ; exit with carry clear
          JR NZ, LL6        ; if>≈2.3 ms, is a 1
          CP 86             ;
          JR NC, LL3        ; if <≈0.8 ms, is noise
LL6:      CCF               ; here if data bit: carry set
                            ; if 0, clear if 1
          RL (HL)           ; shift bit into dest
          DEC E
          JR NZ, LL3
          CALL ENDBYT       ; go get next bit
          JR LL27           ; here after 8 bits
LL7:      DEC D
          JP P-O            ; if have been through ENDBYT
          DEC (I Y+E - LINE ; else restore E-LINE
            +1 - 4000h)
          JR LS8
```

Save

The first thing the SAVE routine does is to POP a value off the stack. This is the address of the instruction after the CALL to SAVE. Since SAVE does not ever return there (it either goes to NEWLIN [the section of the ROM that gets a new line of Basic] or to NEW), it does not want to leave the stack cluttered with its RETurn address.

By listening to a SAVEd program on tape, you will notice that the first element of a SAVEd program is a 5 second silence. This is generated by the next section of code. The DE register pair is loaded with 4811 (decimal), the number of times to execute the "silence loop."

The silence loop runs for 1.04 microseconds (millionths of a second) each time around, and its algorithm is as follows: First, load the keyboard mask to read the BREAK key. (See *SYNC* 1:3, "The ZX80 Keyboard," for details on the ZX80's keyboard system.) Next, read the keyboard/cassette input port. This has the effect of resetting, to zero, the output to the cassette port. The RRA instruction, at LSAV0 is used to rotate bit 0 of the input register, A, into the carry flag. This bit represents the BREAK key data. Then, if NC (No Carry) is true, the routine jumps to LS8, the abort section of the code, because BREAK was pressed. The DJNZ instruction is used to create a delay. Finally, the loop counter, DE, is decremented and compared with 0. If it is zero, then SAVE has executed the loop 4811 times (i.e. ≈ . 5 seconds). Otherwise, the computer will proceed back, to LSAV2, and execute the loop again.

Now we are ready to SAVE RAM (Random-Accesss Memory). Here things start to get a bit hairy. The next section of code begins by picking up the start address for the SAVE, 4000h, and putting it in the HL register pair, the address pointer. The character loop begins here. It starts by setting D=8 (i.e., the number of bits in a byte). Now, set E=248 (i.e., the loop constant for the inter-bit gap [see later]).

The bit-save loop starts next (LSAV3). It proceeds as follows: First, execute an RLC (HL) instruction. This rotates the memory location pointed to by HL, left via the carry flag (i.e., everything shifts left one bit; bit 7 moves into the carry flag; bit 7 moves into bit 0). This operation is depicted in Figure 1.

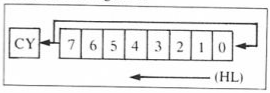


Figure 1. The RLC (HL) instruction.

Notice that the bit-save loop executes 8 times, so the computer cycles through all the bits of the source once, and returns the location pointed to by HL to its original value when done with that byte. Next, execute an SBC A,A (SuBtract with Carry) instruction. In a nutshell, this instruction says: set A=A-A-carry; i.e., A=-1 if the bit, held in carry, is a 1, and A=0 if the bit is a 0. (N.B., -1d =FFh. Also -1d =1111 1111b.) So now we have A=0000 0000b if the bit we are SAVEing is a 0, otherwise A=1111 1111b. Execute an AND 5 instruction. This sets A equal to 0 or 5. Now, ADD A,4 (add 4 to A). Finally copy A into C with an LD C,A instruction.

At this point C=9, if we are SAVEing a 1 bit, and C=4 if we are SAVEing a 0 bit. This number is the loop constant for the "tone loop." Basically, the tone loop's algorithm is:

1) Set cassette output=on (Create a tone).
2) Delay.
3) Set cassette output=off. Read the BREAK key.
4) Delay.
5) Decrement C (the number of times to execute the tone loop).
6) If C≠0 then go back to step 1.

For a full understanding of how the hardware generates a tone, a detailed study of the ZX80's schematic diagram is required. That is, however, beyond the scope of this article.

We continue the bit-save loop, by loading B with E (the inter-bit gap loop constant), and performing NOP (NO oPeration) instructions. (The DJNZ instruction is a special Z80 looping instruction. It says: decrement B and JR (Jump Relative) to some arbitrary memory location (LSAV5, in this case) if B≠0.) Finally, decrement D, the number of bits in this byte, to SAVE and jump back to LAV3 if D≠0.

If you look back and review the tone loop, you will notice that step three not only resets the cassette output, but also reads the BREAK key. This leaves bit 0 of A=0 if BREAK was pressed. The next two instructions, RRA (Rotate Right Accumulator) and JR NC,LS8, move bit 0 of the accumulator into the carry flag, and jump to LS8 if NC (i.e., BREAK was pressed).

SAVE nears completion by CALLing the subroutine ENDBYT, which increments HL, the pointer to the byte to SAVE, and, if there is more memory to SAVE, i.e., HL < (E-LINE), it returns. If ENDBYT finds there is no more memory to SAVE, it jumps to NEWLIN, which is the main section of the 4K Basic. If ENDBYT RETurns to SAVE, the instruction after the CALL ENDBYT, a JR LSAV27, causes the computer to go back and SAVE another byte.

Load

LOAD also begins with a POP operation, used to remove its return address from the stack, thus insuring clean operation. At LL1, DE is loaded with 22290d. This is the loop constant for the leader loop. The leader loop proceeds as follows:

1) Read the keyboard/cassette port.
2) If BREAK is pressed, jump to LS8.
3) If cassette port = one then go back to step 1.
4) Decrement DE (the loop constant).
5) If de≠0 (i.e., not 5 secs. of silence yet) then go back to step 1.

In essence, this loop searches for about 5 seconds of silence on tape. If a one is found during this time, it goes back and resets DE, the loop counter, thus reinitiating the 5 second search.

If BREAK is presed during this period, the LOAD routine jumps to LS8, which, in turn, jumps to NEWLIN. This allows you to change your mind about LOADing a program without deleting the program currently in memory. Hitting BREAK after the first byte has been transferred from cassette into RAM, however, is a destructive action.

Now we are almost ready to start LOADing into RAM. The final pre-LOAD step is to increment the high byte of E-LINE. This is done, in case this byte is 40h, to assure that ENDBYT permits at least 256 bytes to be loaded. Once this first page (256 bytes) of memory is LOADed, E-LINE, a system variable which resides in the first page of RAM, will have been overwritten with the appropriate value.

The routine at LL7, which is where LOAD jumps if BREAK is pressed after the silence loop finishes, tests to see if LOAD has been through ENDBYT, i.e., at least one byte has been LOADed. If it finds that LOAD has not called ENDBYT yet, then it permits the BREAK to be non-destructive. After decrementing the MS (Most Significant) byte of E-LINE back to its original value, it jumps to LS8, and thus back to NEWLIN. If ENDBYT has been called, the LS8 routine jumps to location 0, effectively resetting the computer.

Finally, we can begin to LOAD data into RAM. LOAD continues by loading HL, the destination byte-pointer, with 4000h (16384d) (i.e., the address of the first RAM location). Next, E is set to 8: the number of bits in a byte. The character loop begins with this instruction (LL27). The bit loop begins next (LL3).

The first four instructions of the bit loop are used to read the cassette/keyboard port and, if BREAK is pressed, jump to LL7. Next, two RLA (Rotate Left Accumulator) instructions are executed to rotate the cassette data into the carry flag. If the carry flag is clear (i.e., the cassette port was at logic 0), then LOAD jumps back to LL3. This section of code, by reading the cassette port every 16 microseconds, waits until 1 is found. This 1, which represents a tone burst, marks the beginning of a bit frame.

As soon as 1 is found, the loop falls through, and C is set to 148d. C is used to count the sum of the durations of each pulse and their trailers found during this bit frame. When the bit loop finally falls through, after the LL5 instruction, the C register is checked to see how long the input was on. If bit 7 of C is reset (i.e., C>≈2.3 ms), the JRNZ,LL6 instruction jumps to the routine at LL6 which shifts the bit into the destination. Otherwise, if C < 86 (i.e., C<≈.8 ms) the input was noise, so go back to LL3 and continue looking for the next bit. If the JR NZ, LL3 instruction (used to test for noise) falls through, the LL6 routine executes.

In LL6 an RL(HL) instruction is used to rotate the bit found on the cassette port into the destination. This is followed by a DEC E instruction and a JR NZ,LL3. This pair is used to go back and get the remaining bits for this byte. When E=0, the current byte has been filled, the bit loop falls through, and ENDBYT is called. If ENDBYT comes back, a JR LL27 is executed and the computer goes back for the next byte.

Reviewing the LOAD and SAVE routines, you will notice in each, an LD HL,4000 instruction. This instruction loads HL, the byte pointer, with 16384d (4000h), the address of the first memory location to SAVE or LOAD. Also, looking at ENDBYT will reveal that the system variable E-LINE holds the address of the last-byte-to-SAVE (or LOAD) plus 1. These two values, the 4000h in the LD instruction, and the contents of E-LINE, can be modified to allow any area of memory to be

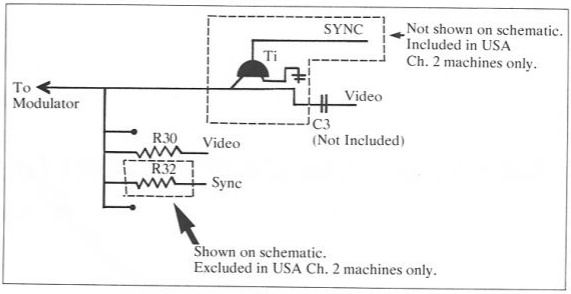


Figure 2.
The above correction applies to USA machines running on channel 2. It does *not* apply to channel 36 ZX80 kits. The transistor is used, in place of R32, to pull the composite video signal, running into the video modulator, low during sync time.

SAVEd or LOADed. My modification to these routines, to make the SAVE/LOAD area fully programmable, is to change the LD HL,4000h to an LD HL,x, where x is the address of the first memory location to SAVE from or LOAD to. Then, each time before you use the routines, POKE the appropriate value into E-LINE. The second part is easy. The first, however, is impossible (or almost). The difficulty lies in the fact that the LOAD/SAVE routines, and therefore, the LD HL,4000h instructions they contain, are in ROM. And you *cannot* change a ROM!

There is, however, a solution: copy the routines into RAM. Then, when you want to use the routines to SAVE/LOAD some section of memory, just POKE the start address into the two byte address field of the LD HL,xxxx instruction. Next, POKE the address of the first byte *not* to SAVE/LOAD (i.e., the last-byte-address + 1) into E-LINE. Then, instead of using the LOAD or SAVE commands from Basic, use a USR(x) call to execute your modified versions of cassette handling routines in RAM.

This method of utilizing the 4K ROM's SAVE and LOAD routines has many possible applications. You can SAVE and LOAD not only independent programs and data, but any area of memory. This means, for example, that you could write a program to selectively SAVE/LOAD some arbitrary (group of) variable(s). You could write a subroutine, for your text-editing system, that SAVEd a string array (i.e., a list of lines of text). You could then write a corresponding LOADer routine for your PASCAL compiler, which retrieved the text from tape. You could write a machine language monitor/debugger/editor (MDE) which allowed you to SAVE/LOAD machine language programs in any format you desire.

This capability of independent data storage has been missing from the ZX80's repertoire for too long. I hope this article will give you the basics required to apply this powerful technique. If the reader response is great enough, some particular applications could be developed in a future column. Something I am working on, also perhaps for a future column, is a full fledged I/O processing system. It could handle reading and writing of records (blocks of data) in a cassette system. With a little bit of electronics, it could even control the motors in your tape recorder.

Video Sync Signals

Some owners have told me that they have tried my direct video circuit to drive a monitor (see *ZX80 Technical Manual/SYNC* 1:2, p. 12), apparently to no avail. The problem seems to be that some monitors require external sync sources. Vertical sync can be obtained from IC11/pin22 (for negative-going signals) and IC11/pin10 (for positive-going signals). Horizontal sync can be tapped from IC19/pin6 and IC19/pin5 for positive-going and negative-going pulses, respectively. These signals must usually be provided to separate terminals. Some monitors provide internal syncing as a selectable option. If yours does, use it.

Video Modulator Drive Circuitry

If you have a standard U.S. version ZX80, *running on channel 2 (VHF)*, then your computer includes some extra circuitry for driving the video modulator. This information is not supplied with the ZX80 schematic. The circuit is shown in Figure 2.

The circuitry is used to superimpose negative-going sync pulses onto the video signal. Your schematic probably shows R2 used for this purpose. However, it is done this way only in the U.K. (*channel 36, UHF*) machines. R32 is not included in U.S. Channel 2 machines.

Loading

As you probably know, if not from experience, then by reading about it, the LOADing problem has been one of the greatest difficulties associated with ZX80s. In my dealings with machines, however, I have found that most of the units returned with a claim of "I CAN'T GET IT TO LOAD!!!!" work fine. Therefore, the assumption I must make is that the procedures being followed are incorrect, or, not being followed correctly. I will detail everything that Sinclair and I, as an individual, have learned about LOAD problems. The following six items are relevant to the situation.

1) Try cleaning and demagnetizing the record/playback heads of the tape recorder. Your ear may not pick up the signal fluctuations due to magnetic flux on the tape heads. The ZX80, however, has a much more sensitive "ear."

2) Use a *good* quality tape. A "DATA TAPE" works well.

3) Check the output level of your tape deck's EAR/SPKR jack. It *must* be *at least* 4 volts peak-to-peak (5 to 6v p-p is best). If you do not have a signal with enough amplitude, the ZX80 will continue looking for your program forever, and the TV screen will remain indeterminently gray indefinitely.

4) Do not use the output from a hi-fi amplifier, as this may damage your computer.

5) Try loading with only the EAR jack connected.

6) A DIN-type socket usually gives only about 1.5 volts peak-to-peak (or less). Therefore, it is unsuitable for use with your ZX80, unless equipped with an external buffer circuit.

Well, that's it for this issue. Until next time. Same relativistic time period. Same non-Euclidian universe. ■

8K ROM
16K RAM

GRA+PIX: A System for Pixel Graphics

Robert B. Keller

The following article describes a set of subroutines for drawing lines, arcs, regular geometric shapes, and sectors on the Sinclair ZX80 with the 8K Basic ROM and 16K RAM. The routines should also work on a ZX81 with 16K RAM. In addition, in SLOW mode on the ZX81, you can watch the figures develop as they are drawn, although for some shapes, such as "filled" circles, this will be a very slow process as many points must be calculated before they can be plotted on the display.

The routines presented are useful for displaying data pictorially as pie charts, histograms, and line charts. They are not suitable for animation, so no attempt has been made to optimize them for speed. Also it is left to the reader to provide labeling and titling routines. This is a chore that is best tailored to the individual application, and it should not be difficult to do.

The routines require a full D-FILE to be effective, thus smaller memory systems could not use the package as is. Those users with 8K of RAM (see *SYNTAX ZX80*, March 1981) should be able to use the entire package. Those with 4K of RAM will have to delete the driver (lines 1 through 255) and the fill routine (lines 9750 through 9895) to fit memory and supply the required parameters by assigning the variables directly before calling the GRA+PIX subroutines. You will have to experiment to see how useful this is.

The term graphic means literally to form by writing, drawing, or engraving—a product of graphic art. Graphics is the art of representing an object on a two-dimensional surface by means of some mathematical rules of projection.

The least rigorous interpretation of a graphic representation is then as simple as a list of words or numbers on a piece of paper set there to convey some meaning to the intended audience. Thus all computer programming languages have some degree of graphic capability. They are all capable of displaying information, either on a hardcopy device, such as a printer or terminal, on a display surface, such as a CRT, or in some other fashion so that the recipient of the information is able to understand what the computer and the program did for him.

Robert B. Keller, 1004 Skyline Drive, Rochester, MN 55901.

With the advent of Basic and the various time-sharing and microcomputer Basic implementations, computer capability has been put within reach of large numbers of people. As these people have become familiar with using the computer, they have demanded and received enhancements in the language. One significant area is the graphic capability of the Basic language. Text graphics is no longer sufficient. With the ability to create moving displays and the capability to form characters by merely setting an electron beam on or off, it became desirable to create picture graphics.

The various microcomputers offer similar ways to do this. With commands such as SET, PLOT, and DRAW, the user is given the ability to turn on or off picture elements (pixels) on his video display in any order by specifying some coordinate point, x-y, and stating whether the point is to be turned on or turned off. Once could laboriously construct any image he wished (within the resolution of the screen) by turning on all the pixels he needed to make the desired image.

However, this can be quite tedious and the novelty of using graphics would quickly vanish if there were not a better way. Fortunately, there is an easier method for many applications of picture graphics. Consider drawing a straight line from the point x=10, y=10 to the point x=50, y=50. One way to do this would be the following simple program.

```
10 LET Y=10
20 FOR X=10 TO 50
30 PLOT X,Y
40 NEXT X
```

The result of this program is a straight line from (10,10) TO (50,10). We could change the vertical displacement of the line by changing line 10:

```
10 LET Y=20
```

and we would now have a straight line from (10,20) to (50,20). We could even ask for the value of y to be input:

```
10 PRINT "INPUT Y"
11 INPUT Y
```

and we could draw horizontal lines all over the screen. So far not very exciting, but it gets better. In a similar fashion we could give a single value of x, allowing y to vary and PLOT vertical straight lines. But how about lines at any angle, from any point (X0,Y0) to any other point (X1,Y1)?

Let us throw away the simple straight line program and brush up a bit on our high school geometry. Look at your Basic manual in the section describing PLOT and UNPLOT.

Note how the screen is divided for these commands. The screen consists of small cells, or pixels, in the horizontal (x) and vertical (y) directions, counting from x=0 to x=63 and y=0 to y=43, with the origin at (0,0) in the lower left-hand corner of the screen. This looks just like the graph paper we used to use in high school.

Remember quadrants? If we look at the graph paper, or screen in this case, and if we take a point in the center, say (x=30,y=20), and draw a vertical and horizontal axis line through that point, we would have the screen divided up into four sections, or quadrants. The upper right quadrant is quadrant I, the upper left quadrant II, the lower left quadrant III, and the lower right quadrant IV. See Figure 1.

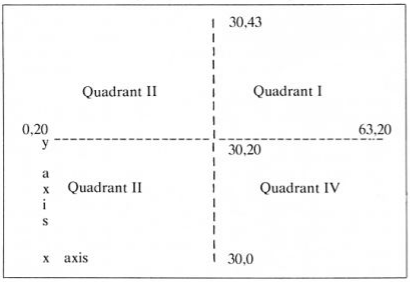


Figure 1.

Now if we draw a straight line from the origin of this graph paper, (30,20), to another x-y point, (50,20) we have a horizontal line lying on the x-axis of the first quadrant. Suppose we draw the line from (30,20) to (30,40). We have a straight vertical line lying on the y-axis between the first and second quadrant, at 90 degrees (one quarter of a circle) to the first line. Now draw a line from (30,20) to (50,40). This is still a straight line, but now it lies half way between and x and y axes, at a 45 degree angle. Without getting too rigorous (or boring) in our mathematics, we can develop a relationship between the line we want to draw and the x and y coordinate points using a little geometry.

Draw a triangle with the base a straight line from (30,20) to (50,20). Draw a vertical line from (50,20) to (50,40). Now connect the top of the vertical line (50,40) to the origin (30,20), and we have our triangle. The diagonal line is the one we want to express geometrically. See Figure 2.

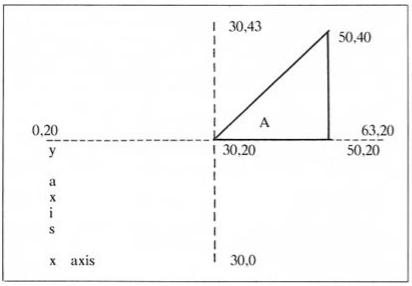


Figure 2.

Between the lines (30,20) to (50,20) and (30,20) to (50,40) is an included angle. Let us call it angle A. We can solve for the value of this angle (forget that we already know it is a 45 degrees or PI/4 radians) by using the relations:

```
SIN A  =  Opposite side / Hypotenuse
       =  (40-20)/SQR((40-20)**2+(50-30)**2)
       =  20/SQR(800) = 0.707106....
```

The angle whose SIN (ASN) is 0.707106..... is 45 degress or PI/4, which we already knew (in this case). The equation for SIN A can be generalized:

```
SIN A = (Y1-Y0)/SQR((Y1-Y0)**2+(X1-X0)**2)

A = ASN((Y1-Y0)/SQR((Y1-Y0)**2+(X1-X0)**2))
```

Now that we have A, so what? Well, for one thing we can now solve for the line we want to draw from (30,20) to (50,40) in this case, or better, from (X0,Y0) to (X1,Y1) in general. Using geometry the line we want (the hypotenuse of our triangle) is related to the tangent of the angle A we just calculated. The tangent is merely the ratio of the SIN to the COS, TAN A = SIN A/COS A, and can be used in the following manner.

For values of x from X0 to X1, in steps of 1, the value of y is calculated as:

```
DY = SGN (Y1-Y0)*TAN A
Ycurrent = Yprevious + DY
```

Thus, treating x as the independent variable we are able to calculate the value of each point along the desired line geometrically. Listing 1 is a routine for drawing a straight line from any point (X0,Y0) to (X1,Y1). The routine takes into account that you may have division by zero possibilities (drawing a point) and optimizes the line for the fit. The quadrant is taken into account. The missing lines will be supplied later. The variable P is the PLOT/UNPLOT control. If P=0, the routine will UNPLOT.

For each routine the SYNCSUM is supplied so you can check your entry as you go (see *SYNC* 1:4, pp. 6-7). Begin by entering the LINE routine in a NEW workspace. Enter the lines as they are listed, making the line deletions when asked to do so. Each SYNCSUM is obtained by examining the contents of memory at the time, so if you want to check your SYNCSUM with that on the listing, it is important that lines be entered in sequence. Also, do not use tokens in REM statements to save space but spell words out exactly as listed so the correct SYNCSUM is obtained.

```
9000 REM LINE (X0,Y0,X1,Y1,P)
9015 IF X0<>X1 OR Y0<>Y1 THEN GO
TO 9040
9020 LET D=2
9025 LET DX=0
9030 LET DY=0
9035 GOTO 9105
9040 REM CHECK SLOPE OF LINE
9045 LET A=ABS ((Y1-Y0)/SQR ((Y1
-Y0)*(Y1-Y0)+(X1-X0)*(X1-X0)))
9050 LET A=ASN (A+(1-A)*(A>1))
9055 IF A>PI/4 THEN GOTO 9085
9060 LET D=2*(ABS (X1-X0)+1)
9065 LET DX=SGN (X1-X0)
9070 LET DY=SGN (Y1-Y0)*TAN A
9075 IF ABS DY<1E-7 THEN LET DY=
0
9080 GOTO 9105
9085 LET D=2*(ABS (Y1-Y0)+1)
9090 LET DX=SGN (X1-X0)*COS A/SI
N A
9095 IF ABS DX<1E-7 THEN LET DX=
0
9100 LET DY=SGN (Y1-Y0)
9105 LET X2=X0-DX
9110 LET Y2=Y0-DY
9120 FOR I=1 TO D/2
9125 LET X2=X2+DX
9130 LET Y2=Y2+DY
9135 IF X2<L1 OR X2>U1 THEN GOTO
 9155
9140 IF Y2<L2 OR Y2>U2 THEN GOTO
 9155
9145 IF NOT P THEN PLOT X2,Y2
9150 IF P THEN UNPLOT X2,Y2
9155 REM HOLD PLACE FOR NOW
9170 NEXT I
9175 RETURN

                    SYNCSUM = 40
```

Listing 1.

Lines 9135 and 9140 perform a useful function. Normally the full screen will be used for plotting, but on some occasions you may wish to restrict some area for text only, without any possibility of an intrusion from a line calculation. If the variables L1,L2,U1,U2 are set to 0,0,63,43 then the full screen will be used for plots. L1,U1 are the lower and upper limits of the x-axis; and L2, U2, the lower and upper limits of the y-axis. If you would like to restrict the area for plotting to the right half of the screen for example, then set L1=31, U1=63, L2=0, U2=43. If this is done, then any point calculated outside the right half of the screen will not plot. The points will still be calculated, however.

The switch from TAN A to COT A (COS A/SIN A) at PI/4 is done to avoid the division by zero for situations where you have vertical or near vertical lines. The signum (SGN) function is used to adjust to the proper quadrant.

At this point you may ask why such overkill for drawing a simple line? You would be right—a line can be calculated in somewhat simpler fashion. A great deal of care is being taken with this, however, as the LINE routine will be used as a basis for plotting many geometric shapes, such as arcs, polygons, and segments. It will also be used to fill solid objects, so exact boundary conditions must be maintained. For now let us review what is needed to use the routine just presented. Listing 2 will serve as a driver program:

```
 1 LET L1=0
 2 LET L2=0
 3 LET U1=63
 4 LET U2=43
 5 LET P=0
10 CLS
15 PRINT "INPUT X0,Y0: ";
20 INPUT X0
25 PRINT X0;
30 INPUT Y0
35 PRINT " ";Y0
40 PRINT "INPUT X1,Y1: ";
45 INPUT X1
50 PRINT X1;
55 INPUT Y1
60 PRINT " ";Y1
65 PRINT "HIT N/L TO CONTINUE"
70 INPUT A$
75 CLS
80 GOSUB 9000
85 INPUT A$
90 GOTO 10
```

Listing 2.

In Listing 2 you are prompted for the starting point, X0,-Y0, and the ending point, X1-Y1. You could also have requested whether or not you would like to change the screen limits, set by lines 1-4.

Suppose instead of drawing a line from X0-Y0 to wherever we instead used X0-Y0 as the center of some shape we want to draw. Take for example a circle. If we let X0-Y0 be the center and call the hypotenuse of the previous discussion the radius of the circle, then we should be able to calculate a whole series of X1-Y1 points lying on the perimeter of the circle by varying the angle A from 0 radians (or degrees) to 2*PI radians (360 degrees). If we then connected these points with straight lines we could approximate the shape of a circle. The more points we calculated, the better the circle would look.

Call the origin point 01 and 02 rather than X0, Y0 to avoid ambiguities. Let our convention be to draw the circle starting with A=0 radians, counter-clockwise through quadrants I - IV, until A=2*PI radians. The starting x,y point would then be:

X0=01+R
Y0=02

It can be shown that the x coordinate varies with the COS A and the y coordinate varies with the SIN A:

X1=01+R*COS A
Y1=02+R*SIN A

where R is the length of the radius (hypotenuse). Furthermore if we divide the circle up into N sections, or segments, then

we can work around the perimeter of the circle in multiples of A radians:

1<=i<=N

A = 2*PI/N

then

Xi=01+R*COS(i*A)

Yi=02+R*SIN(i*A)

Now we can develop a program to draw circles using the line program as a subroutine. Delete lines 1 to 90 of the Listing 2 driver program to get a good SYNCSUM. Listing 3 is a circle example.

```
9300 REM CIRCLE (01,02,R,N,P)
9305 LET A1=2*PI/N
9310 LET X0=01+R
9315 LET Y0=02
9320 FOR J=1 TO N
9325 LET X1=01+R*COS (J*A1)
9330 LET Y1=02+R*SIN (J*A1)
9335 GOSUB 9000
9340 LET X0=X1
9345 LET Y0=Y1
9350 NEXT J
9355 RETURN
                    SYNCSUM = 32
```

Listing 3.

A simple driver program to use the circle program is found in Listing 4.

```
  1 LET L1=0
  2 LET L2=0
  3 LET U1=63
  4 LET U2=43
  5 LET P=0
 10 CLS
 15 PRINT "ENTER ORIGIN: 01,02:
"
 20 INPUT 01
 25 INPUT 02
 30 PRINT "ENTER RADIUS: R"
 35 INPUT R
 40 PRINT "NO. SEGMENTS/CIRCLE:
N"
 45 INPUT N
 50 CLS
 55 GOSUB 9300
 60 INPUT A$
 65 GOTO 10
                    SYNCSUM = 134
```

Listing 4.

Try playing with this function for a while. RUN it and see what happens as N is varied from 1 and 2 and larger. Notice that all the figures you see plotted are circles. Since the PLOT pixels are relatively coarse, lines at any angle other than vertical or horizontal are rather ragged, and some peculiar things may happen because of this. Ignoring this, you should be seeing what look like triangles, squares, pentagons, and so on until N is large enough that the polygon begins to look like a circle. In fact, it is never a circle but always a polygon due to the coarseness of the pixels, but it does approximate a circle.

Instead of using an angle A let us break the angle up into two angles, PHI (P2) and THETA (T2) and generalize the circle function a bit. If we take a polygon and draw it on our graph paper, we could rotate it counter-clockwise on its origin at any angle from 0 to 2*PI radians. After 2*PI radians the cycle repeats. We will call this angle the inclination angle, PHI (P2).

The included angle can also vary from 0 to 2*PI radians. If the angle is 2*PI radians, then the object is enclosed and we shall call it a polygon—a "generalized circle." If the angle is less than 2*PI radians, then the object is not closed and we have an arc. Call the angle THETA (T2). A generalized routine is shown in Listing 5. There are some calculations that may not seem necessary, but they will be used later. Delete the Listing 4 driver program lines 1 to 65 to ensure a good SYNCSUM. The polygon routine will overlay the previous circle program if entered correctly, so these lines need not be expressly deleted.

```
9300 REM POLYGON/ARC (O1,O2,R,N,
T2,P2,P)
9305 LET F=1
9310 GOSUB 9900
9315 REM ENTER FROM SEGMENT
9320 LET XO=INT (P5+O1+R*COS P2)
9325 LET YO=INT (P5+O1+R*SIN P2)
9330 FOR J=1 TO N1
9335 IF F THEN GOTO 9345
9340 IF D2>=D1-1 THEN GOTO 9415
9345 LET T4=J*T1
9350 IF ABS (T2-T4)<=0.01 THEN L
ET T4=T2
9355 IF J=N1 AND T3=2*PI THEN LE
T T4=T3
9360 LET X1=INT (P5+O1+R*COS (P2
+T4))
9365 LET Y1=INT (P5+O2+R*SIN (P2
+T4))
9370 LET X3=X1
9375 LET Y3=Y1
9380 GOSUB 9010
9385 IF NOT F THEN GOSUB 9750
9390 LET XO=X3
9395 LET YO=Y3
9400 IF T2=T4 THEN GOTO 9410
9405 NEXT J
9410 IF F THEN GOSUB 9970
9415 RETURN


9900 REM UTILITY
9905 LET T1=(2*PI)/N
9910 LET P2=ABS P2
9915 LET T2=ABS T2
9920 IF P2>2*PI THEN LET P2=  -2
*PI*INT (P2/(2*PI))
9925 IF T2>2*PI THEN LET T2=  -2
*PI*INT (T2/(2*PI))
9930 LET LO=1
9935 LET P3=P2
9940 LET T3=T2
9945 LET P5=0.5
9950 LET NO=INT (P5+T2/T1)
9955 LET N1=NO
9960 LET T2=NO*T1
9965 RETURN
9970 LET P2=P3
9975 LET T2=T3
9980 DIM X(1)
9985 DIM Y(1)
9990 DIM Z(1)
9995 RETURN
                    SYNCSUM = 231
```

Listing 5.

Be certain to include all the lines above, as they will be used. Lines 9900 on are the start of the utility routines that will be used more extensively. Essentially what we have so far is a line program, a polygon/arc routine, some utility functions, and what appears to be some unused code. In the interest of brevity I will not provide a driver program until all the code is complete. It would be a good idea to SAVE your entries on tape often, in case you have problems, so you have some recovery point. I have found that with the 16K RAM attached tape reliability suffers for long files. So a good quality tape, a tape demagnitizer, frequent head cleaning and demagnitizing, several SAVEs in a row to assure a good SAVE, and lots of patience are needed to make LOADable tape copies.

A segment can be visualized as looking like a slice of pie. It is really nothing more than an arc of less than 2*PI radians with two straight lines from the point of origin connecting each end of the arch. We can easily plot a segment by calling the LINE routine to draw the lines from the origin to the arc end points and the polygon routine to draw the arc itself. The routine to do this follows. The variable, F, will be used later when we fill the segments. For the time being assume F to be 1. Listing 6 shows the SEGMENT routine:

Listing 6.

```
9500 REM SEGMENT (O1,O2,R,N,T2,P
2,P,F)
9505 GOSUB 9900
9510 IF T2<=PI OR F THEN GOTO 95
35
9515 LET N1=INT (N/2)
9520 LET LO=INT (0.55+NO/N1)
9525 LET NO=NO-N1
9530 LET T2=N1*T1
9535 FOR L=1 TO LO
9540 LET XO=INT (P5+O1)
9545 LET YO=INT (P5+O2)
9550 LET X1=INT (P5+O1+R*COS P2)
9555 LET Y1=INT (P5+O2+R*SIN P2)
9560 GOSUB 9010
9565 IF F THEN GOTO 9595
9570 LET DO=D
9575 DIM X(D)
9580 FOR J=1 TO D
9585 LET X(J)=Z(J)
9590 NEXT J
9595 LET X1=INT (P5+O1+R*COS (P2
+T2))
9600 LET Y1=INT (P5+O2+R*SIN (P2
+T2))
```

A GOSUB to 9500 with F=1 and the other variables set as desired will plot a segment on the screen. The remaining task is to fill the segment (or unfill it) if desired. By fill, I mean to use the outline of the segment drawn and turn on or off all the pixels inside its boundaries. In this way pie charts can be created.

This procedure requires two steps: first to draw the outline of the shape and "remember" it, and second to uniquely turn on each point within the shape. It is a cycle burner to say the least, as you will discover when you run the routine. A filled circle (a segment with the included angle T2 equal to 2*PI radians) with a radius of 20 and number of segments 24 will take somewhat less than 5 minutes in FAST mode! You can enjoy several cups of coffee if you watch it in SLOW mode on a ZX81.

The approach I have used is quite simple, but, still quite time consuming. Using a value of F=0 for fill/unfill, the line routine is triggered to generate a vector of the x-y points it creates. When the return is made to the segment routine, it stores the vector Z in a new vector X or Y, depending on which edge it is. LINE is called again, repeatedly, to generate the vector of arc points. After each call, one segment at a time, each point in the arc vector is connected uniquely to the corresponding point in the X or Y vector by a call to the LINE routine. The line drawn is either a vertical or a horizontal line, from a point in Z to a corresponding point in either vector X or Y as appropriate. If there are not enough points in Z to finish the fill, an adjustment is made. The angle of inclination, PHI (P2), is checked to minimize redundant or non-connecting points.

```
9605 GOSUB 9010
9610 IF F THEN GOTO 9645
9615 LET D1=D
9620 LET D2=1
9625 DIM Y(D1)
9630 FOR J=1 TO D1
9635 LET Y(J)=Z(J)
9640 NEXT J
9645 GOSUB 9315
9650 IF F THEN GOTO 9715
9655 IF D0=2 THEN GOTO 9695
9660 LET D=D1
9665 DIM Z(D)
9670 FOR J=1 TO D-1 STEP 2
9675 LET Z(J)=Y(D-J)
9680 LET Z(J+1)=Y(D-J+1)
9685 NEXT J
9690 GOSUB 9750
9695 IF N0<N1 THEN LET N1=N0
9700 LET N0=N0-N1
9705 LET P2=P2+T2
9710 LET T2=N1*T1
9715 NEXT L
9720 GOSUB 9970
9725 RETURN          SYNCSUM = 168
```

It should be noted that to make life a little easier, the included angle, THETA (T2), is forced to be an integer multiple of the size of the segment angle, N. You may set it to whatever you wish but it will be adjusted if necessary. This should not be too noticeable given the coarseness of low resolution pixel graphics.

A second note in using the system is this: segments are the only objects that will be filled. If you want to fill a polygon, just call the segment routine with a value for THETA (T2) of 2*PI radians and you can create a filled polygon (or unfill an existing filled polygon). It is that simple. Listing 7 shows the remainder of the subroutine code for GRA+PIX.

```
9005 LET F=1
9115 IF NOT F THEN DIM Z(D)
9155 IF F THEN GOTO 9170
9160 LET Z(2*I-1)=INT (X2+P5)
9165 LET Z(2*I)=INT (Y2+P5)


9750 REM FILL ROUTINE
9755 LET F=1
9760 LET P4=1
9765 IF (P2>=0 AND P2<P5*PI) OR
(P2>=PI AND P2<(1+P5)*PI) THEN L
ET P4=0
9770 FOR K=1 TO D-1 STEP 2
9775 LET X0=Z(K)
9780 LET Y0=Z(K+1)
9785 IF P4 THEN GOTO 9805
9790 IF X0<>X1 THEN GOTO 9820
9795 LET Y1=Y0
9800 GOTO 9880
9805 IF Y0<>Y1 THEN GOTO 9820
9810 LET X1=X0
9815 GOTO 9880
9820 IF D0=2 THEN GOTO 9855
9825 IF K<>1 THEN LET D0=D0-2
9830 IF NOT P4 AND X0<>X(D0-1) T
HEN GOTO 9820
9835 IF P4 AND Y0<>X(D0) THEN GO
TO 9820
9840 LET X1=X(D0-1)
9845 LET Y1=X(D0)
9850 GOTO 9880
9855 IF K<>1 THEN LET D2=D2+2
9860 IF NOT P4 AND X0<>Y(D2) THE
N GOTO 9855
9865 IF P4 AND Y0<>Y(D2+1) THEN
GOTO 9855
9870 LET X1=Y(D2)
9875 LET Y1=Y(D2+1)
9880 GOSUB 9010
9885 NEXT K
9890 LET F=0
9895 RETURN          SYNCSUM = 158
```

Listing 7.

```
  1 REM GRA+PIX DRIVER MODULE
  5 DIM S$(64)
 10 LET R1=PI/180
 15 LET L1=0
 20 LET L2=0
 25 LET U1=63
 30 LET U2=43
 35 LET S=0
 40 PRINT AT 20,0;S$
 41 PRINT AT 20,0;"CHANGE L1,L2
,U1,U2 FROM: ";L1;" ";L2;" ";U1;
" ";U2;" (Y/N)?"
 45 INPUT A$
 50 IF A$="N" THEN GOTO 65
 55 PRINT AT 20,0;S$
 56 PRINT AT 20,0;"ENTER L1,L2,
U1,U2:"
 60 INPUT L1
 61 INPUT L2
 62 INPUT U1
 63 INPUT U2
 65 PRINT AT 20,0;S$
 66 PRINT AT 20,0;"PLOT OR UNPL
OT? (P/U):"
 70 INPUT A$
 75 IF A$="P" THEN LET P=0
 80 IF A$<>"P" THEN LET P=1
 85 PRINT AT 20,0;S$
 86 PRINT AT 20,0;"LINE,POLYGON
,ARC, OR SEGMENT?   (L/P/A/S):"
 90 INPUT A$
 95 IF A$="L" THEN GOTO 225
100 IF A$="P" THEN GOTO 160
105 IF A$="A" THEN GOTO 135
110 LET S=1
115 PRINT AT 20,0;S$
116 PRINT AT 20,0;"FILL (OR UNF
ILL)? (Y/N):"
120 INPUT A$
125 IF A$="Y" THEN LET F=0
130 IF A$<>"Y" THEN LET F=1
135 PRINT AT 20,0;S$
136 PRINT AT 20,0;"ENTER INCLUD
ED ANGLE (T2) AND   ORIENTATION
ANGLE (P2) (DEGREES):"
140 INPUT T2
141 INPUT P2
145 LET P2=P2*R1
150 LET T2=T2*R1
155 GOTO 180
160 LET T2=2*PI
165 PRINT AT 20,0;S$
166 PRINT AT 20,0;"ENTER ORIENT
ATION ANGLE (P2)    (DEGREES):"
170 INPUT P2
175 LET P2=P2*R1
180 PRINT AT 20,0;S$
181 PRINT AT 20,0;"ENTER ORIGIN
(O1,O2):"
185 INPUT O1
186 INPUT O2
190 PRINT AT 20,0;S$
191 PRINT AT 20,0;"ENTER RADIUS
(R):"
195 INPUT R
200 PRINT AT 20,0;S$
201 PRINT AT 20,0;"HOW MANY SEG
MENTS/FULL CIRCLE?"
205 INPUT N
210 IF S THEN GOSUB 9500
215 IF NOT S THEN GOSUB 9300
220 GOTO 240
225 PRINT AT 20,0;S$
226 PRINT AT 20,0;"ENTER X0,Y0,
X1,Y1:"
230 INPUT X0
231 INPUT Y0
232 INPUT X1
233 INPUT Y1
235 GOSUB 9000
240 PRINT AT 20,0;S$
241 PRINT AT 20,0;"ANOTHER SET?
(Y/N):"
245 INPUT A$
250 IF A$<>"N" THEN GOTO 35
255 STOP
                    SYNC SUM = 114
```

Listing 8.

The GRA+PIX routines are normally expected to be used as subroutines to another program, however they can be used from a driver program, calling the routines over and over and building images to gain some experience with how the routines and pixel graphics work. The driver routine in Listing 8 will prompt you for the required parameters, plot the desired figure, and return to you to ask for more input while leaving the image just plotted on the screen. If you do not like the image plotted, just unplot it by responding to the driver prompts. You can in this way build up complex geometric shapes on the screen. Your imagination is the limit, so have fun!

Some of the code in the GRA+PIX program had to be written the way it appears because of some problems in the 8K Basic ROM. I think I have programmed around most of these errors, but there may be some cases I have not discovered.

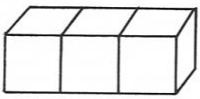
You need not be limited to low-res graphics. It should be possible to use your own character set to create high resolution graphics characters by creating your own 8x8 bit pattern and calling the video driver from RAM rather than ROM. This is one reason I have elected to use the TAN/COT approach for generating lines rather than the simpler delta x/delta y step method. If time permits, I will attempt to follow up with a method to do this.

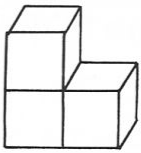
I would be interested in hearing comments on the use of GRA+PIX, and would be happy to try to answer any questions if a SASE is enclosed. ■

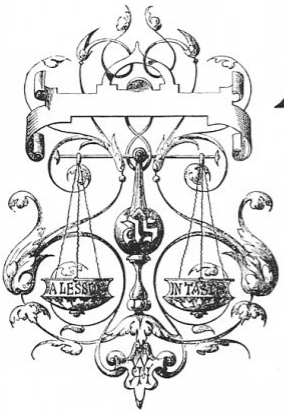
puzzles & problems

The Play School Problem

We're going back to school for this puzzle, way, way back. It has been noted that, if you take three wooden cubes, or blocks, and glue them squarely face-to-face, only two different configurations are possible (see figures at the right). In our problem the reader is given four cubes to work with. How many different shapes can you construct with this many cubes? Please don't get stuck on your first try!





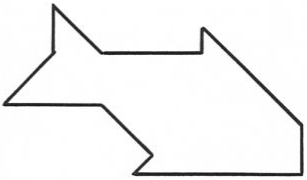


The False Scales

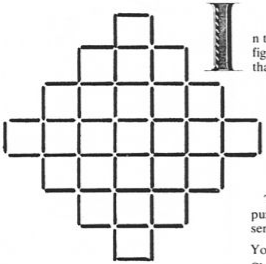
A cheese put into one of the scales of a *false* balance was found to weigh 16 lbs. When placed in the opposite scale it weighed 9 lbs. only. What was its actual weight? (From *Merlin's Puzzler* by Charles Barry Townsend; published by Hammond, Inc.)

The Shark Problem

Certainly one of the most enduring forms of mechanical puzzles in the world is the Tangram. It has been around for hundreds of years. Below is pictured an oblong set of Tangram tiles. Next to it is the outline of an oriental fighting fish. The puzzle, of course, is to rearrange the seven Tangram tiles into the shape of this fish. This can be done by lightly drawing lines in the fish where you think the pieces should go. Don't get hooked now; it's not as easy as it looks.



A Matchless Problem



In this puzzle you must first arrange 64 matches into the figure pictured at the left. This figure contains 25 squares. Now comes the hard part. You must remove 32 matches so that you are left with just 6 squares. Can you do this in less than 5 minutes?

The Two Numbers

There are two numbers, such that twice the first *plus* the second equals 17, and twice the second *plus* the first equals 19. Find the numbers. (From *Puzzles Old & New* by Professor Hoffmann, circa 1890).

That's all until next issue, folks. I hope that you enjoyed the above problems. If you have a puzzle that you would like to share with our readers, please send it along. If Merlin uses it, he will send you a copy of one of his famous books.

Your editor,
Charles Barry Townsend

Answers on page 23.

4K ROM
8K ROM
1K RAM

The PEEK Function and the POKE Command

Dr. I. S. Logan

Introduction

The first three articles in this series have all involved machine code language programming. This article returns to Basic to discuss the PEEK function and the POKE command. Since the 8K ROM is now widely available, both the 4K ROM and the 8K ROM are included in the discussion and program illustrations. Although this is written for the "average" reader, I hope that the information will be useful to the "advanced" reader as well.

The PEEK Function

In the operating manual the PEEK function is described as a function that gives the value of the byte in memory for a given address. The use of the PEEK function is demonstrated in Programs 1 and 2. We suggest that you run the PEEK Demonstrator Program at this point.

```
 10 PRINT "PEEK DEMONSTRATOR"
 20 PRINT
 30 PRINT "ENTER ADDRESS (0-655
    35)"
 40 INPUT ADDRESS
 50 PRINT
 60 PRINT "LOCATION <";ADDRESS;
    "> CONTAINS:"
 70 PRINT
 80 LET A=PEEK(ADDRESS)
 90 PRINT "DECIMAL#<";A;">"
100 PRINT "HEX#####<";CHR$(A/16
    +28);CHR$((A AND 15)+28);">"
110 IF A=118 THEN GO TO 130
120 PRINT "CHR$#####<";CHR$(A);"
    >"
130 PRINT "BINARY##<";
140 LET B=128
150 FOR C=1 TO 8
160 IF A<B THEN GO TO 200
170 LET A=A-B
180 PRINT "1";
190 GO TO 210
200 PRINT "0";
210 LET B=B/2
220 NEXT C
230 PRINT ">"
240 INPUT A$
250 CLS
260 RUN
```

Program 1: PEEK Demonstrator (4K ROM; 1K RAM)

Dr. I. S. Logan, 24 Nurses Lane, Skellingthorpe, Lincoln, LN6 OTT, U.K. This article is the fourth in a series.

```
 10 PRINT "PEEK DEMONSTRATOR"
 20 PRINT
 30 PRINT "ENTER ADDRESS (0-655
    35)"
 40 INPUT ADDRESS
 50 PRINT
 60 PRINT "LOCATION <";ADDRESS;
    "> CONTAINS:"
 70 PRINT
 80 LET A=PEEK ADDRESS
 90 PRINT "DECIMAL#<";A;">"
100 PRINT "HEX#####<";CHR$ INT
    (A/16+28);CHR$ INT (A-INT (A/16)
    *16+28);">"
110 IF A=118 THEN GOTO 130
120 PRINT "CHR$#####<";CHR$ A;">
    "
130 PRINT "BINARY##<";
140 LET B=128
150 FOR C=1 TO 8
160 IF A<B THEN GOTO 200
170 LET A=A-B
180 PRINT "1";
190 GOTO 210
200 PRINT "0";
210 LET B=B/2
220 NEXT C
230 PRINT ">"
240 INPUT A$
250 CLS
260 RUN
```

Program 2: PEEK Demonstrator (8K ROM; 1K RAM)

Next we must consider the terms 'memory' and 'address.' Figure 1 shows a simplified view of how the Z80 (in either a ZX80 or a ZX81) is joined to 'memory' by a DATA BUS and an ADDRESS BUS.

The ADDRESS BUS is a track that carries 16 binary signals in parallel from the Z80 microprocessor to the RAM and the ROM. (RAM random accss memory; ROM read only memory.)

Because the ADDRESS BUS of a Z80 system is able to carry 16 binary signals, it is possible to uniquely address 65536 locations in memory.

The addresses for these locations are:

Binary	0000 0000 0000 0000 to 1111 1111 1111 1111
Decimal	0 to 65535
Hex.	0000 to FFFF

In the ZX80 and ZX81 systems the operand for the PEEK function must be an address in this range. Most of the locations will give an 'echo' of some other location as much of the possible 64K memory is not actually used.

The DATA BUS is a track that carries 8 binary signals in parallel back and forth between the Z80 and the memory. In the case of the ROM, the signals can only pass towards the Z80, whereas in the case of the RAM, the signals can pass in either direction.

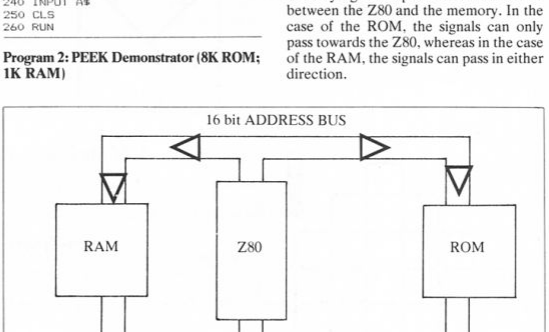


Figure 1.

Every location in the memory is also capable of holding 8 binary signals and therefore in a Z80 system a 'byte of data' can be represented by 8 binary digits which will have the ranges:

Binary	0000 to 1111
Decimal	0 to 255
Hex.	00 to FF

The PEEK function will therefore always return a value in the decimal range 0 to 255.

The Execution of the PEEK Function

Let us now consider how the PEEK function is handled by the Basic interpreter in the 4K and the 8K monitor programs. First, take the case of the execution of the simple program:

4K	8K
PRINT PEEK(0)	PRINT PEEK Ø

This will print the decimal contents of the first location in the memory.

For the 4K program the result will be 33, (LD HL,+dddd);
for the 8K,
211. (OUT (+FD),A).

The line scanning part of the Basic interpreter when dealing with this line will execute the PRINT COMMAND ROUTINE since the first token of the line is a PRINT.

In the PRINT COMMAND ROUTINE (4K address: 0972; 8K address: OACF) the next character of the Basic line is considered. In the example above the operand for the PRINT command is the expression PEEK(0) or PEEK 0. This expression is then evaluated, and the result of the evaluation is stored as the 'last value.'

In the 4K interpreter the 'expression evaluator' is found at 09E1 (Hex), and the 'last value' is the system variable pair 16418 and 16419 (Decimal). If the result is numeric, then this pair of locations will hold a 2's complement 16 bit number.

In the 8K interpreter the 'expression evaluator' is found at OF52, and the 'last value' is the area of memory designated as MEM. A numeric result will always be in the form of a 5 byte floating-point number.

In both cases the evaluation of the operand of the PEEK function is itself a subject of an 'expression evaluation' and the forming of a 'last value.'

Once the 'last value' for the expression PEEK(0) or PEEK 0 has been calculated, control returns to the PRINT COMMAND ROUTINE where a call is made to the PRINT LAST VALUE ROUTINE. In the 4K interpreter this routine is at 06F1 (Hex) and the operation of the routine can be demonstrated by using:

RUN USR(1777).

This will print the 'last value' (in this case 1777 Decimal). In the 8K interpreter the routine is at 15D7 but this cannot be demonstrated easily.

The POKE Command

In the operating manual the POKE command is described as being a command that must be followed by two parameters. The first parameter, which itself can be an expression, must give a 'last value' that can be used as an address of a location in the memory. The second parameter, which can also be an expression, must give a 'last value' that is in the decimal range 0-255. Execution of the command will write the value of the second parameter into the location addressed by the first parameter.

```
10 PRINT "POKE DEMONSTRATOR"
20 PRINT
30 PRINT ,"■■■" (3 shift A's)
40 PRINT ,"■ ■" (shift A;
   sp.; shift A)
50 PRINT ,"■■■" (3 shift A's)
60 PRINT
70 PRINT "ENTER A CHARACTER AN
   D NEWLINE,##OR JUST NEWLINE TO S
   TOP"
80 INPUT A$
90 IF A$="" THEN STOP
100 POKE PEEK(16396)+PEEK(16397
    )*256+41,CODE(A$)
110 GO TO 80
```

Program 3: POKE Demonstrator (4K ROM; 1K RAM)

Programs 3 and 4 show values being POKEd into the display file.

```
10 PRINT "POKE DEMONSTRATOR"
20 PRINT AT 2,8;"■■■" (3 shift A's)
30 PRINT TAB 8;"■ ■" (shift A;
   sp.; shift A)
40 PRINT TAB 8;"■■■" (3 shift A's)
50 PRINT AT 6,0;"ENTER A CHARA
   CTER AND NEWLINE,##OR JUST NEWLI
   NE TO STOP"
60 INPUT A$
70 IF A$="" THEN STOP
80 POKE PEEK 16396+PEEK 16397*
   256+41,CODE A$
90 GOTO 60
```

Program 4: POKE Demonstrator (8K ROM; 1K RAM)

In both programs the center of the box is defined as D-FILE +41. In the 8K ROM program line 80 can be replaced by:

80 PRINT AT 3,9;CHR$ CODE A$

which uses the PRINT AT routine at 0918 in the 8K monitor program to calculate the required address.

The Execution of the POKE Command

The syntax for the POKE command is:

POKE (expression) (,) (expression)

and this is checked by referral to the syntax table.

In the 4K ROM the appropriate entry is seen in Listing 1 and in the 8K ROM in Listing 2.

In the 4K ROM the appropriate entry is:

Address	Contents	Comment
07A4	06	The first expression is of type '6'.
07A5	D8	The 'comma'.
07A6	05	The second expression is of type '5'.
07A7	D1	The address of the POKE COMMAND
07A8	09	ROUTINE is 09D1.

In the 8K ROM the appripriate entry is:

Address	Contents	Comment
0C80	06	The first expression is of type '6'.
0C81	1A	The 'comma'.
0C82	06	This time, again type '6'.
0C83	00	No further requirements.
0C84	92	The address of the POKE COMMAND
0C85	0E	routine is 0E92.

```
                                        Comment
 10 RANDOMISE                           A good idea.
 20 GO TO 110                           Jump past display.
 30 PRINT "PELMANISM"                   Title
 40 PRINT
 50 PRINT "HERE ARE THE CARDS"
 60 PRINT
 70 PRINT C$                            The cards.
 80 PRINT "1.2.3.4.5.6.7.8."            The numbers of the cards.
 90 PRINT
100 RETURN
110 LET C$="? ? ? ? ? ? ? ?"            Eight cards; face-down.
120 DIM A(8)                            Value of cards.
130 LET P=0                             Pair counter initialised.
140 FOR B=38 TO 41                      Will be cards A,B,C & D.
150 FOR C=1 TO 2                        Form a pair.
160 LET D=RND(8)                        A card.
170 IF NOT A(D)=0 THEN GO TO 16
    0                                   Already assigned?
180 LET A(D)=B                          Assign value to card.
190 NEXT C                              Next of pair.
200 NEXT B                              Next value.
210 GO SUB 30                           PRINT BOARD.
220 GO SUB 370                          INPUT ROUTINE.
230 LET G=F                             Copy number of first.
240 GO SUB 370                          INPUT ROUTINE
250 IF F=G THEN GO TO 230               They were the same!
260 IF NOT A(G)=A(F) THEN GO TO
320                                     Are they a pair?
270 LET B=-1+PEEK(16392)+PEEK(1
    6393)*256                           Find the card in C$,
                                        in the variable area.
280 IF PEEK(B+G*2)=0 THEN GO TO
320                                     Pair already chosen?
290 POKE B+G*2,0                        Remove the cards from
300 POKE B+F*2,0                        the board.
310 LET P=P+1                           Score the pair.
320 PRINT P;"PAIRS FOUND, PRES
    S N/L"                              The status report.
330 IF P=4 THEN GO TO 420               All over?
340 INPUT A$                            Wait for N/L.
350 CLS
360 GO TO 210                           Back for next try.
370 PRINT "CARD#";                      INPUT ROUTINE
380 INPUT F                             The chosen card, 1-8.
390 IF F<1 OR F>8 THEN GO TO 38
    0                                   Validity test.
400 PRINT F;"#IS A -#";CHR$(A(F
    ))                                  The value of the card.
410 RETURN                              Return with card F.
420 CLS
430 PRINT#,"WELL DONE"                  The point of success.
```

Readers who have larger memories might like to increase the number of cards on the board by changing lines 80, 110, 129, 140 (increase 41 to reflect the number of pairs), 160, 330 (number of pairs), and 390.

Program 5: Pelmanism (4K ROM; 1K RAM)

In the POKE COMMAND ROUTINEs the actual POKEing is done with the following lines:

	Address	Menmonic
4K:	09DF	LD (DE),A
8K:	0EA5	LD (BC),A

Note that in both cases only a single byte is loaded into a memory location.

Here is a simple game that uses the features discussed in the main body of the article.

Pelmanism

This game is usually played with a pack of playing cards. The cards are placed face-down on a table and the player, or players in turn, take two cards and look at them. If the cards are of the same value, then the player scores 1. If the cards are different, then they are replaced.

The game is therefore one of memory. In the programs 5 and 6 below the cards are kept in string C$, and cards that have already been found to be pairs are removed from the string using a POKE operation.

The author would be pleased to see any programs that have been written following the ideas mentioned in this article.

```
 10 RAND                                Comment
 20 GOTO 80                             A good idea.
 30 PRINT "PELMANISM"                   Jump past display.
 40 PRINT AT 2,0;"HERE ARE THE          Title.
    CARDS"
 50 PRINT AT 4,0;C$                     The cards.
 60 PRINT "1.2.3.4.5.6.7.8"             The numbers of the cards.
 70 RETURN
 80 LET C$="? ? ? ? ? ? ? ?"            Eight cards, face-down.
 90 DIM A(8)                            Value of cards.
100 LET F=0                             Pair counter initialised.
110 FOR B=38 TO 41                      Will be cards A,B,C & D.
120 FOR C=1 TO 2                        Form a pair.
130 LET D=INT (RND*8+1)                 A card.
140 IF A(D)<>0 THEN GOTO 130            Already assigned?
150 LET A(D)=B                          Assign value to card.
160 NEXT C                              Next of pair.
170 NEXT B                              Next value.
180 GOSUB 30                            PRINT BOARD.
190 GOSUB 340                           INPUT ROUTINE
200 LET G=F                             Copy number of first.
210 GOSUB 350                           INPUT ROUTINE
220 IF F=G THEN GOTO 200                They were the same!
230 IF A(G)<>A(F) THEN GOTO 290         Are they a pair?
240 LET B=1+PEEK 16400+PEEK 164         Find the card in C$,
    01*256                              In the variable area.
250 IF PEEK (B+G*2)=0 THEN GOTO         Pair already chosen?
290
260 POKE B+G*2,0                        Remove the cards from
270 POKE B+F*2,0                        the board.
280 LET P=P+1                           Score the pair.
290 PRINT P;"#PAIRS FOUND, PRES         The status report.
    S N/L"
300 IF P=4 THEN GOTO 400                ALL OVER?
310 INPUT A$                            Wait for N/L.
320 CLS
330 GOTO 180                            Back for next try.
340 PRINT
350 PRINT "CARD#";                      INPUT ROUTINE
360 INPUT F                             The chosen card, 1-8.
370 IF F<1 OR F>8 THEN GOTO 360         Validity check.
380 PRINT F;"#IS A -#";CHR$ A(F         The value of the card.
    )
390 RETURN                              Return with card F.
400 CLS
410 PRINT TAB 8;"WELL DONE"             The point of success.
```

Program 6: Pelmanism (8K ROM; over 1K RAM)

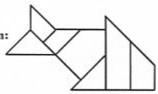
Note that in this program the first card is at:
"1+PEEK 16400+PEEK 16401*256"

puzzle answers

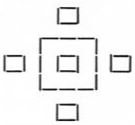
The Play School Problem: Eight different shapes are possible.

The False Scales: The answer is 12 lbs. Problems of this class are solved by ascertaining the square root of the product of the two weights. In this case 9 x 16 = 144, and the square root of 144 is 12, the required answer.

The Shark Problem:



A Matchless Problem:



The Two Numbers: The required numbers are 5 and 7. For if twice the first + the second = 17, and twice the second + the first = 19, then the sum is 36. This is the same as three times the first + three times the second. The sum of the numbers themselves must be 36 ÷ 3 = 12. Since twice the first + the second is an odd number, the second is also an odd number, and the first, being the even number 12 less an odd number, must also be an odd number. Now the only pairs of odd numbers which together make 12 are 1 and 11, 3 and 9, 5 and 7. Of these, we find by experiment that 5 and 7 are the only two that answer the conditions: 5 x 2 + 7 = 17, and 7 x 2 + 5 = 19.

4K ROM
1K RAM

Machine Language Teaches the ZX80 to READ

Edward A. Kennedy, Jr.

The Need for a READ

READ is a very useful statement. FORTRAN has it, and so do the Hewlett-Packard Basics. Microsoft, the company that wrote the Basics for most of the small American computers, has provided READ statements in its languages. However, the ZX80 does not have it. This is the one statement that I have missed more than any other in the Sinclair 4K Basic.

INPUT works fine if the program needs different data every time it is RUN. For example, a program to balance your checkbook would use INPUT to get the information about the check you wrote today. But many programs require large quantities of unchanging data in their arrays before they do their things. A lot of game programs fall into this category and so do some industrial control programs. This is normally the job of the READ statement.

In a game program, for instance, an array would be created by a DIMension statement, and then the necessary data would be READ into each element. In Basic it would look something like this:

```
10 DIM A(50)
20 FOR J=0 TO 50
30 READ A(J)
40 NEXT J
50 DATA 174,39,317,255,78,131,
286,228,224,152,158,186,247,241,
85,161,24,145,50,271,38,106,165,
95,313,206,95,261,80,58,259,296,
24,1,178,133,268,41,249,250,279,
294,66,323,179,115,81,66,93,200,
281
```

Listing 1.
A READ statement as it might look on the Sinclair if it had one.

This part of the program would first open space in memory to store an array named "A" and having fifty-one storage locations (from A(0) through A(50) inclusive). It would then initialize A(0) with the value 174, A(1) with 39, A(2) with 317, and so on until it placed 281 into A(50).

Of course, there are ways to get around the READ deficiency. We could eliminate line 50 and change line 30 to:

```
30 INPUT A(J)
```

Then we would hand our friends a list of fifty-one numbers to type in each time that they wanted to play the game. Somehow, I think this would kill much of the fun! Furthermore, this way is sensitive to errors. Sometimes a program can be completely spoiled by a single error.

Edward A. Kennedy, Jr., 16 701 Red Oak St., Bensenville, IL 60106.

A better substitute for our missing READ statement would be:

```
20 LET A(0)=174
30 LET A(1)=39
40 LET A(2)=317
50 LET A(3)=255
60 LET A(4)=152
and so on until:
510 LET A(49)=200
520 LET A(50)=281
```

Listing 2.
Initializing an array without a READ statement.

This would work! The data would be permanently in the program and the possibility of errors occurring each time you ran it would be eliminated. It has two disadvantages. It requires a lot of typing, and it ties up two-and-a-half to three times as much memory as would have been required if we had the READ statement to work with (as in Listing 1). The program of Listing 1 would require 221 bytes assuming that DATA and READ were one-byte keywords. On the other hand, the program of Listing 2 would require 644 bytes to get the same fifty-one numbers into the same fifty-one array locations.

So I decided to create a subroutine that would function a lot like the READ statement does in the native machine language of the Z-80 CPU (Central Processing Unit), and I could get at it with the USR function, when I needed it.

Using the USR

The ZX80 has a "USeR" function that tells the machine to do a machine language subroutine beginning at any place in memory that the programmer chooses. Suppose that you had a machine subprogram starting at memory address 14,336. Also suppose that this subroutine had to give us back some information. You might write the following:

```
210 ...
220 LET INFO=USR(14336)
230 ...
```

When the machine went to execute line 220, it would first run the USR subroutine and give back a number.

The Z-80 CPU has many internal locations in which it can store numbers that it is working on. We call these internal storage cells "registers." When the computer comes back from running the subroutine, it would still have one more thing to do with line 220. It would have to copy the number in two of its registers into the variable named INFO. The new value of INFO would be equal to the value of the H and L registers (taken as a pair) *at the instant that the machine language subroutine was completed.*

This means that when I wrote the READ subroutine, I had to make sure that the number it was READing got into the HL register-pair before the Z-80 found the instruction to RETurn from the subroutine to the execution of the Basic line that called it. Keep this in mind if you plan to write subroutines of this type.

The programs in Listings 3 and 4 show what the USR function looks like from the programmer's point of view.

```
1 REM---------
10 POKE 16430,201
60 LET INFO=USR(16430)
70 PRINT "THE VALUE OF USR
=",INFO
```

Listing 3.
USR returns the number 16430.

Line 1 is the REMark line where the machine code will be stored. The first thing that happens when you run this program is that line 10 POKEs 201 into a location in line 1. That number, 201, is the whole subroutine! It is the decimal form of the instruction telling the Z-80 to RETurn.

After line 10 puts the instruction into memory, line 60 tells the computer to go to the exact same spot and do whatever commands it finds there. But the Z-80 finds RET at the first location. So it makes

no changes whatever; it simply comes back to the Basic program. Then it takes whatever number it finds in the HL register and puts a copy of it into the variable named INFO. When RUN, this program gives:

THE VALUE OF USR= 16430

This is the number that was in the HL register! It is also the number of the location where the subroutine starts. This tells me that the computer probably uses the HL register to help it get to the subroutine in the first place.

Now when I look at the program again, I notice that line 1 has changed. One of the dashes has changed to a question mark. That is because the POKE changed the *contents* of one of the memory locations being used to store line 1. Before we ran the program, that cell held number 220. When the computer looks up 220 in its table, it finds that it must display a dash. However, when it looks up the 201 that we put there instead, it finds that no display character has been assigned yet. So it jumps to an error routine and prints a question mark. There are twenty-five different numbers which can produce that question mark when POKEd into line 1. Typing a question mark is equivalent to POKEing the number 15. So if I want one of the others, I must use the POKE statement as I did here.

One of the most important things to understand about this method is that a number stored in that location can mean different things to the system at different times. That same 201 can be a low level instruction to the processor, or it can be a number that must be interpreted and transferred to the display file. Under different circumstances, it might even have been a statement in Basic that the machine would have to look up and execute. In each case the action is different.

Let me expand that program a little! I have not shown that the value of the HL register-pair becomes the value of USR. The following lines added to Listing 3 begin to do that.

```
 1 REM---?------
10 POKE 16430,33
20 POKE 16431,0
30 POKE 16432,0
40 POKE 16433,201
60 LET INFO=USR(16430)
70 PRINT "THE VALUE OF USR
   =",INFO
```

Listing 4.
USR will equal the value in 16431 plus 256 times the value in 16432, unless 16432 is more than 127 in which case USR will be negative.

The Z-80 recognizes the decimal number 33 (which line 10 will now place in memory) as the instruction to put the number that follows it in its L register, and the one after that in its H register. (When taken together, L is the lower part and H is the higher part of the pair.)

If you RUN it now, you will be told:

THE VALUE OF USR= 0

Zero is the number we told it to put in the HL pair! If you doubt it, change line 20 to POKE different numbers at that location. If you use values from 64 through 127, however, you stand a strong chance of confusing your computer so thoroughly that you might lose your program. For the moment, you will probably want to stick to the values from 0 through 63, and from 128 through 255.

I was fascinated by the changes in line 1 when I put different codes in it. The "Character and Keyword Appendix" of the user's manual gives details of what each number stands for.

Supporting the Effort

By now, you can see that programming in machine code on the ZX80 is not the easiest thing in the world. Typing in all those POKE statements is difficult enough for a subroutine that occupies only four bytes, so imagine what it would be like for a program that filled 159 bytes as our READ subroutine does!

A Word about Hexidecimal

Another difficulty is that each variation of each instruction that the Z-80 has corresponds to one or more numbers. In order to know which number(s) I need, I have to look at a table. These tables are written in either a binary number code or a hexidecimal number code. I find the hex tables easier to use.

Since the ZX80 requires a decimal number code for its POKEs and returns a decimal number when it does a PEEK, it is up to us to do some converting.

The hexidecimal number system has sixteen possible numbers that can be put in any position:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, & F

Remember that A, B, C, D, E, and F are not letters here; they are numbers just like 8 and 9 are numbers.

For instance, the hexidecimal form of the instruction LD HL,NN, used in Listing 4 to get a number into HL, is 21. That is not twenty-one because it is not two tens and a one. It is two sixteens and a one, or the same as thirty-three in decimal.

The hexidecimal table lists the number for RETurn as C9. This means that there are C times sixteen plus nine times one. Since C equals 12, multiply 12 by 16 and add 9:

(12x16) + (9x1) = 201

In Listing 4 you will see that line 10 POKEs 33 and line 40 POKEs 201. These are exactly the values just computed.

Making It Automatic

When only three or four numbers are involved, this is not too demanding, but doing 159 numbers is another matter. Fortunately, a program can be written in Basic to eliminate the need for all those POKEs and conversions. Listing 5 gives such a program that will run on 1K. While it looks short, it runs into trouble if it tries to display all 159 bytes of the machine program. So, after you type in the 105th entry, it takes a moment to erase the screen so that you have room to finish the subroutine.

If you are running the program and want to stop, type S and then NEWLINE. To start again, type CONTinue and NEWLINE. When you do, the program will PEEK at all the locations that were on the screen before you stopped, convert them to hex, and display them again. So you will pick up exactly where you left off.

If, on the other hand, you find that you made a mistake five bytes back, you will first need to stop the program as above. Then you type in:

LET I=I-5

followed by NEWLINE. Then, when you start the program again, you will be five spaces back and ready to make the needed correction. The same is true if you want to look at the next twenty locations. Stop the program, add twenty to "I", and then press CONTinue and NEWLINE.

When you are entering code with this program, you must type in both numbers. The ZX80 will not supply a missing zero. If you want 00, you must type 00 and not 0. A single zero will be entered as the hex number E5, and that is not the same at all.

Conclusion

In this article I have discussed what a READ statement does and why it is handy. The subroutine given above will duplicate the action of the READ statement, but it must be stored in a REMark to protect it and to allow us to SAVE it. The USR function is the means of retrieving the material in storage. Finally, a 1K program enables you to put material into the storage.

```
  1 REM --------------------------  ;160 minus signs reserve space for machine
--------------------------------  ;     codes
--------------------------------  ;     23 in first line
--------------------------------  ;     32 in each of the next four lines
--------------------------------
--------                          ;     and 9 in the sixth line
  2 DIM U(1)                      ;Machine subroutine uses U(0) for address
  3 LET U(0)=16600                ;     of next DATA and U(1) as a flag
  5 LET R=16427

 50 FOR I=R TO 16585              ;Start and end of machine code storage
 60 IF I-R=105 THEN GO TO 160     ;Go make room in the display for the rest
 70 INPUT C$                      ; Enter machine code
 80 LET C=CODE(C$)
 90 IF C=56 THEN GO TO 150        ;If it was an "S" then go to "STOP"
100 IF C<28 OR C>43 THEN GO TO    ;If not hexidecimal go back and try again
70
110 PRINT C$;" "; (2 sp.)         ;Else display it with two spaces after it
120 POKE I,16*(C-28)+CODE(TL$(C   ;Then convert and POKE it
$))-28
130 NEXT I
140 GO TO 999                     ;Avoid running over

150 STOP
160 CLS                           ;Portion of routine displays codes that
170 LET S=R                       ;     are actually stored in memory
180 IF I>R+104 THEN LET S=R+80    ;If too many locations will be displayed
190 FOR J=S TO I-1                ;     delete first 80 and display the rest
200 LET K=PEEK(J)                 ;Look at the number in this location
210 LET L=K/16                    ;Convert it and
220 PRINT CHR$(L+28);CHR$(K-L*    ;     convert some more and display it
16)+28;" "; (2 sp.)               ;     do not forget the two spaces
230 NEXT J
240 GO TO 70
999 STOP
```

Listing 5.

Short (1K) version of Basic program accepts and converts hexidecimal machine code and stores it in its own line 1. (Enter "S" to STOP!) Hex codes between 40 and 7F can scramble the system when put in a REMark line, and must be avoided unless lines 5 and 50 are changed to store the codes where they will not be displayed.

To anticipate, Part 2 will present the READ subroutine and explain more about the "forbidden" codes, methods of getting along without them, and some of the features of the system.

For readers interested in more detailed study, I suggest the following works:

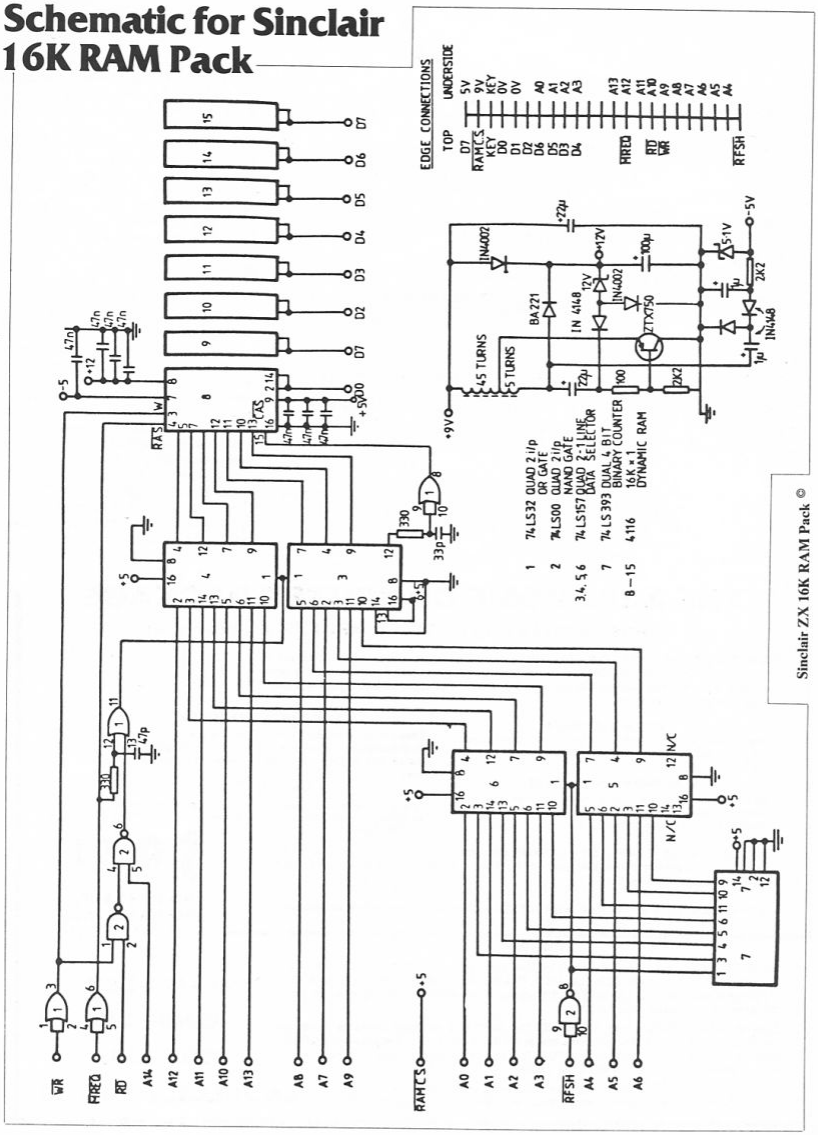
Barden, William, *Z-80 Microcomputer Design Projects,* Indianapolis: Howard Sams & Co., 1980. This book has me excited. It shows how to make a manual EPROM programmer and a small dedicated computer with software for several projects.

Barden, William. *The Z-80 Microcomputer Handbook,* Indianapolis: Howard W. Sams & Co., 1978. Good for its clarity and organization.

Nichols, Elizabeth A.; Nichols, Joseph C.; and Rony, Peter R. *Z-80 Microprocessor Programming and Interfacing.* 2 books. Indianapolis: Howard W. Sams & Co., 1979. Book 1 has the most complete tables. Both volumes fill certain other gaps. They give a number of experiments that apply to an Italian computer.

Zaks, Rodney, *How to Program the Z-80.* Berkley, CA: Sybex, 1979. This is a 624 page monster, but it seems exceedingly clear. It devotes an incredible 284 pages to the Z-80 instruction set. ■

Schematic for Sinclair 16K RAM Pack



Using Key and Token Expressions

Richard W. McDaniel

While translating a TRS-80 program into a ZX80 program, I crashed the system. I had already saved most of the program, so I loaded it again and proceeded to cut anywhere possible to save memory. When the program was as compact as I could get it, I ran it again. After a few inputs the program stopped. I quit for the day.

A couple of days later, I was writing directions for a game in a REM statement and accidentally pressed the shift key and the "/" key simultaneously. Instead of "/", "NOT" appeared! I experimented more with this new technique and discovered that keywords as well as tokens could be typed into program lines in full—spaces and all—with practically a single keystroke.

Richard W. McDaniel, Box 71, Glasgow, VA 24555.

This technique not only saves typing time, but, because a keyword or a token is usually stored as a single byte, it also saves memory. I went back to the program I had been translating and modified it with this technique. It ran perfectly.

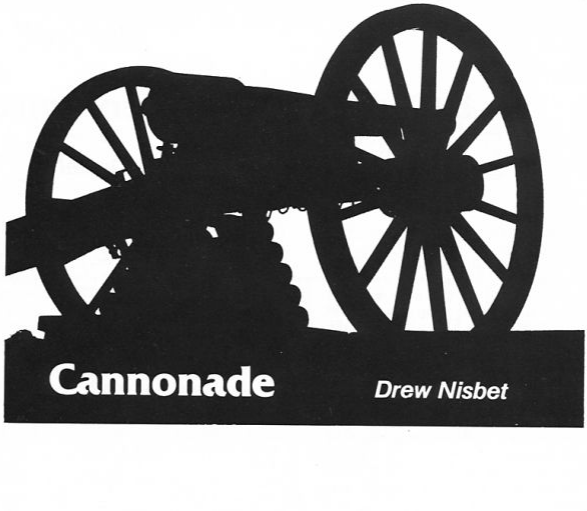
Let us look at some examples of how the technique works.
The program line:
10 REM TO RUN, USE GOTO 100
written the ordinary way takes 24 bytes whereas with the above key and token technique it only takes 14 bytes, for a saving of 10 bytes. A line such as:
20 PRINT "ENTER YOUR NAME"
can be
20 PRINT "INPUT YOUR NAME"
for a saving of 4 bytes.

In a line like
30 LET Z$="JIM AND JOE"
you save 8 bytes by using the token "AND".

To use the technique in a line such as
100 PRINT "TO STOP PROGRAM, INPUT S"
type the statement number. Next type the last keyword first; then back up using shift "5" and enter the next to last keyword and so on until all keywords are entered. After that, type the keyword that uses the keyworded-characterstring either REM, PRINT or a characterstring, then type the tokens in their respective places. Finally, type any alphanumerics. The technique used in the above line saves 9 bytes.

If the keyword or token is preceded by another keyword or token, the preceding space of the following expression is omitted. If there is an alphanumeric between keywords or tokens, the spaces of each remain the same. More information can be found on page 105 of your ZX80 operating manual. I hope you find this technique as useful as I have. ■

4K ROM
1K RAM



Cannonade

Drew Nisbet

In *Cannonade* you are the commander (choose your own rank) of a squad of six men. You have been given as your next objective the capture of an enemy gun emplacement. Your men must outmaneuver the gun's handlers. If they are spotted as they are advancing, they may be fired upon. When a man is hit, he cannot return to his base until the gunner is distracted and fires on another target. Should one of your men breach the fortification protecting the emplacement, the gun is silenced and the position is taken.

Since the game program is long, a temporary program is necessary to initialize certain variables so that the program will run in 1K. Enter the program in Listing 1, RUN it, and then delete lines 10 to 120 inclusively by typing the line number and NEWLINE.

Now you can enter the main program (Listing 2). SAVE it as soon as you have finished entering it. Do not RUN it because RUNning will alter the values stored in the variables initialized by the program in Listing 1. To play the game enter GOTO 100.

The prompt will ask you which man you wish to move. Enter a letter from A to F and NEWLINE. The letter on the screen, corresponding to the man you have chosen to advance on the emplacement, will move forward a random distance and may move either up or down one line on the screen. If he is fired upon and hit,

Drew Nisbet, 6 Moffatt Crt., Toronto, Ont., Canada, M9V 4E1.

```
 10 LET T = 4
 20 LET D = 0
 30 LET X = 0
 40 DIM L(20)
 50 DIM M(6)
 60 DIM P(6)
 70 LET L(20) = 2
 80 FOR I = 1 TO 6
 90 LET P(I) = 1
100 LET M(I) = I
110 IF I > 3 THEN LET M(I) = M(I) + 1
120 NEXT I
```

Listing 1.

```
100 RANDOMISE
110 CLS
120 FOR I = 1 TO 7
130 FOR K = 1 TO 19
140 LET L(K) = 0
150 NEXT K
160 PRINT I;" ";
170 FOR J = 1 TO 6
180 IF NOT M(J) = I THEN GO TO 240
190 IF L(P(J)) = 0 OR L(P(J)) = 2 THEN GO TO 220
200 LET P(J) = P(J) + 1
210 GO TO 190
```

Listing 2.

the letter will appear in inverse video. It will remain this way until another man is fired upon by the gunners; then the man can return to the base. This is shown by the return of the letter to the left side of the screen. When a man has broken through the line on the right side, the gun has been captured. The number of moves required will then be displayed. To end the game before the gun is captured, press NEWLINE when the prompt calls for a letter. The number of moves made up to this point will be displayed.

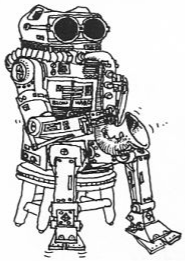
The game is relatively short, but, if you want to increase the difficulty and consequently the playing time, decrement the values which determine the distance moved by changing line 420.

In order to replay the game, you must reload the program from the tape because of the method by which the variables were initialized. ■

Cannonade

```
220 LET L(P(J)) = J + 37
230 IF D = J THEN LET L(P(J)) = J + 165
240 NEXT J
250 FOR K = 1 TO 20
260 PRINT CHR$(L(K));
270 NEXT K
280 LET L(20) = 2
290 IF NOT T = I THEN GO TO 310
300 PRINT "SEE NOTE AT RIGHT";I;    NOTE:
310 PRINT
320 NEXT I                          PRINT CHARACTERS FOR
330 PRINT                           LINE 300 ARE:
340 FOR I = 1 TO 6
350 IF P(I) = 20 THEN GO TO 620       1. SHIFTED "G"
360 NEXT I                            2. SHIFTED "A"
370 PRINT "MAN:"                      3. SHIFTED "D"
380 INPUT A$                          4. SPACE
390 IF A$ = "" THEN GO TO 630
400 LET X = X + 1
410 LET Q = CODE(A$) - 37
420 LET P(Q) = P(Q) + RND(5) + 3
430 IF P(Q) > 20 THEN LET P(Q) = 20
440 LET J = RND(3)
450 IF J = 1 THEN GO TO 510
460 IF J = 2 THEN LET K = -1
470 IF J = 3 THEN LET K = 1
480 LET I = M(Q)
490 IF I + K = 0 OR I + K = 8 THEN GO TO 440
500 LET M(Q) = I + K
510 IF RND(3) = 2 THEN GO TO 100
520 LET P(D) = 1
530 LET D = 0
540 LET K = 0
550 FOR I = 1 TO 6
560 IF P(I) < K OR D = I THEN GO TO 600
570 LET K = P(I)
580 LET D = I
590 LET T = M(I)
600 NEXT I
610 GO TO 100
620 PRINT "GUN CAPTURED IN ";
630 PRINT X;" MOVES."
```

4K ROM
1K RAM



Robot Composer

Cecil Bridges

As Richard Forsen has noted (*SYNC* 2:34), a series of tones can be generated by a series of FOR-NEXT loops. However, we are not accustomed to hearing this pitch scale of tones, and the duration of the tones varies inversely with the pitch. The Robot Composer corrects these deficiencies in this pitch scale and generates rhythmic, melodic tunes.

The series of FOR-NEXT loops used to generate tones occupies lines 500 to 559. Each of these loops has a multiplier for N which corrects the tendency for low notes to be longer. The basic unit of duration, N, is set by line 100. Most of the remainder of the program is used to select the pitch, P, from the tone generator. Because some tones made by the generator are not notes in the familiar diatonic scale, lines 418 to 428 skip such tones. The FOR-NEXT loops corresponding to these skipped tones are spacers and are necessary for the generation of the remaining tones in the diatonic scale. Lines 402 and 405 bounce the pitch off the top and bottom of the scale if the limits of the scale are exceeded.

Lines 5, 10, and 490 set the length of the tune and the value of S, an additive variable for the pitch scale, P. The variable S first drops in value and then rises again towards the end of the tune. Because high values of P are associated with low notes, and vice versa, the pitch will initially rise and then fall.

Cecil Bridges, 1248 N. Denver, Tulsa, OK 74106.

```
5   FOR K=1 TO 8
10  LET S=2*ABS(K-5)
17  LET P=S
20  FOR G=1 TO 4
100 LET N=2
150 LET P=P+RND(7)-4
200 FOR J=1 TO 4
205 LET N=N+1
400 LET P=P+RND(7)-4
402 IF P<1 THEN LET P=RND(7)
405 IF P>19 THEN LET P=5+RND(12)
418 IF P=3 THEN LET P=P+1
419 IF P=5 THEN LET P=P+1
420 IF P=6 THEN LET P=P+1
421 IF P=8 THEN LET P=P+1
422 IF P=9 THEN LET P=P+1
423 IF P=11 THEN LET P=P+1
424 IF P=13 THEN LET P=P+1
425 IF P=14 THEN LET P=P+1
426 IF P=16 THEN LET P=P+1
427 IF P=17 THEN LET P=P+1
428 IF P=18 THEN LET P=P+1
485 GOSUB 497+P*3
490 NEXT J
492 NEXT G
496 NEXT K
498 GOTO 5
500 FOR I=1 TO 13*N
501 NEXT I
502 RETURN
503 FOR I=1 TO 13*N
504 NEXT I
505 RETURN
506 FOR I=1 TO 12*N
507 NEXT I
508 RETURN
509 FOR I=1 TO 12*N
510 NEXT I
511 RETURN
512 FOR I=1 TO 11*N
513 NEXT I
514 RETURN
515 FOR I=1 TO 11*N
516 NEXT I
517 RETURN
518 FOR I=1 TO 10*N
519 NEXT I
520 RETURN
521 FOR I=1 TO 10*N
522 NEXT I
523 RETURN
524 FOR I=1 TO 9*N
525 NEXT I
526 RETURN
527 FOR I=1 TO 9*N
528 NEXT I
529 RETURN
530 FOR I=1 TO 8*N
531 NEXT I
532 RETURN
533 FOR I=1 TO 8*N
534 NEXT I
535 RETURN
536 FOR I=1 TO 8*N
537 NEXT I
538 RETURN
539 FOR I=1 TO 8*N
540 NEXT I
541 RETURN
542 FOR I=1 TO 8*N
543 NEXT I
544 RETURN
545 FOR I=1 TO 8*N
546 NEXT I
547 RETURN
548 FOR I=1 TO 8*N
549 NEXT I
550 RETURN
551 FOR I=1 TO 8*N
552 NEXT I
553 RETURN
554 FOR I=1 TO 7*N
555 NEXT I
556 RETURN
557 FOR I=1 TO 7*N
558 NEXT I
559 RETURN
```

Lines 20 and 150 set the length of melodic phrases within the tune and determine within phrases the pitch increment and variability of the pitch transition from one measure to the next. Lines 200 to 400 determine length and rhythmic composition of the measures, as well as the average pitch increment and variability from one note to the other.

In the program as presented, pitch tends to stay generally within the limits of the pitch scale; small changes can be made in the additive variables in lines 150 and 400, but if large changes are made, the pitch will constantly bounce off the scale limits causing the music to be less interesting. A rising pitch in measures may be produced by changing the -4 in line 150 to a -5, and a -3 will produce a pitch decrement between phrases. Substituting a -5 for -4 in line 400 will produce a rise in pitch within measures and -3 will produce a decrement. Of course, pitch increments in phrases can be combined with decrements in measures or vice versa.

Other functions can be substituted for line 10:

```
10 LET S=-1+(K-5)*(K-5)
```

produces a more rapid initial rise and more rapid decline than the original line 10. By subtracting these functions from a constant, they may be inverted. By changing K, G, and J, the length of tune, phrase, and measure can be changed. You will have to make corresponding changes to lines within the FOR-NEXT loops.

Rhythm may be changed by modifying lines 100, 200, and 205. For instance:

```
100 LET N=6
200 FOR J=1 TO 4
205 LET N=N-1
```

produces a happy rhythm. A drum roll sound will be produced at the end of each measure if the limit of J is greater than N. A wide variety of rhythms is possible using these two examples and changing the values in lines 100 and 205 and the limit in line 200.

To produce only a single tune, line 498 may be omitted.

If a piece of insulated wire is laid under the computer and attached to the antenna of an FM radio, the sound will be transmitted without static to the FM receiver. You can now play your music through a high quality music system.

Make changes cautiously, trying out the program between small changes; even renumbering seems to have an effect on tone quality, and introducing more complicated functions seems to make the sound unpleasantly "fuzzy." If you attempt to lengthen the pitch scale, you will change the frequencies of all the notes; and you will no longer have a diatonic scale. A longer pitch scale is possible, but it is necessary to shorten the rest of the program to get it into 1K.

4K(8K) ROM
1K RAM

Examining Prime Numbers: Two Programs

George J. Repicky

The prime numbers are a set of numbers which have as their distinguishing property the fact that they are divisible evenly only by themselves and one. Thus, 2, 3, 5, and 7 are primes, whereas 4, 6, and 9 are not. While 1 would seem to fit the definition, it is not usually considered to be a prime.

Aside from their own interesting property, mathematics teachers introduce prime numbers in the middle grades because they add a bit of puzzle-like fun to the class and because they provide an excellent vehicle for an introduction to factoring. They make an interesting subject for computers both because the testing of a number to see if it is a prime is an excellent application of the computer's ability to perform many calculations rapidly and because they provide a good illustration of the use to which "IF...THEN" and "FOR...NEXT" statements can be put.

The first of the two programs below tests a set of numbers to see which members of the set are primes. The program asks for (and enters with INPUT statements) the beginning and the end of the set, and displays those numbers in the chosen set which are primes. For example, if in response to the opening statement:
ENTER STARTING NUMBER,
you enter 20, and, if in response to the next statement:
ENTER ENDING NUMBER,
you enter 30, the result:
THE PRIMES BETWEEN 20 AND 30 ARE:
23, 29
will be displayed.

George J. Repicky, 49 Roosevelt Ave., Schenectady, NY 12304.

The second program produces the prime factorization of a selected number. The prime factorization of a number is the set of prime numbers which, when multiplied together, produce the number as their product. For example, the prime factors of 72 are 2, 2, 2, 3, and 3, since 2 X 2 X 2 X 3 X 3 = 72. If, in response to the opening statement:
ENTER A NUMBER BETWEEN 2 AND 32,767
you enter 72, the computer will display:
THE PRIME FACTORS OF 72 ARE:
2, 2, 2, 3, 3,

If a prime number is selected for factorization, the computer will display this fact. If, for example, you entered the number 73, the computer will display:
73 IS A PRIME, AND HAS NO PRIME FACTORS EXCEPT ITSELF.

In addition to producing their answers, both programs cycle again upon entering the number 1. The entering of any number other than 1 will stop the program.

Both programs are written for the 4K ROM. Both can be entered together and will run on 1K RAM, although, if both programs are in the computer, you can run out of memory if you choose a particularly wide range of values when running the first. To run the first program simply use the RUN command; to run the second program, you must enter RUN 1010. When running these programs for the first time it is a good idea to use a short range of values for the first or a low number for the second. Wide ranges or high values can take some time to run. Once you are sure the programs are running well, save them on tape, delete the second. If you plan to take a break, try a wide range (say M = 1, N = 500 or 1000). This will take many minutes to run!

Testing a Range of Numbers for Primes

```
 10 CLS
 20 PRINT                                  20, 30 lower the display.
 30 PRINT
 40 PRINT "ENTER STARTING NUMBER"
 50 INPUT M                                50 accepts the starting number
 60 CLS                                    of the set to be tested.
 70 PRINT                                  70, 80 lower the display.
 80 PRINT
 90 PRINT "ENTER ENDING NUMBER"
100 INPUT N                                100 accepts the ending number of
110 CLS                                    the set to be tested.
120 PRINT                                  120, 130 lower the display.
130 PRINT
140 PRINT "THE PRIMES BETWEEN#";M;"#AND#";N
150 PRINT "ARE:#"
160 PRINT
170 IF M=2 THEN PRINT "2,#";               170 insures that 2 will be displayed.
180 FOR J=M TO N                           180 presents each number in turn.
190 FOR K=2 TO J-1                         190, 200 test each possible divisor;
200 LET Q=J/K                              if a divisor is found, 210 drops
210 IF J=Q*K THEN GO TO 240                the number from further consideration.
220 NEXT K
230 PRINT J;",#";                          230 prints only those numbers found
240 NEXT J                                 not to have a divisor.
250 PRINT
260 PRINT
270 PRINT "TO TEST NEW NUMBERS, ENTER 1"   270 to 290 repeat the program
280 INPUT Z
290 IF Z=1 THEN GO TO 10
300 STOP
```

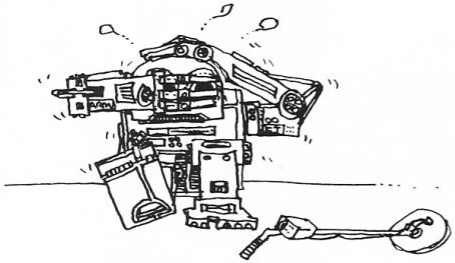
Note: If the 8K ROM is used, line 210 must be replaced by something like:
210 IF J=INT(Q)*K THEN GO TO 240

Prime Factorization

```
1010 CLS
1020 PRINT                                 Lowers display for neatness.
1030 PRINT
1040 PRINT "ENTER A NUMBER BETWEEN         Asks for number to be factored.
     2 AND 32,767"
1050 INPUT N                               Enters number to be factored.
1060 CLS
1070 PRINT                                 Lowers display.
1080 PRINT
1090 IF N=2 THEN GO TO 1180                Identifies 2 as a prime number.
1100 PRINT "THE PRIME FACTORS OF#";
     N;"#ARE:#"
1110 FOR J=2 TO N/2                        Tests possible divisors.
1120 LET Q=N/J                             Defines the quotient.
1130 IF N=Q*J=0 THEN GO SUB 1210           Identifies a divisor which has no remainder.
1140 NEXT J                                Goes to next divisor.
1150 CLS
1160 PRINT                                 Lowers display.
1170 PRINT
1180 PRINT N;"#IS A PRIME, AND             Displays and identifies a number
     HAS NO PRIME FACTORS                  if it is a prime
     EXCEPT ITSELF"
1190 PRINT
1200 GO TO 1280
1210 PRINT J;",#";                         Begin subroutine; prints prime factor.
1220 IF Q=1 THEN GO TO 1260                Determines if all factors have been found.
1230 LET N=Q                               Factors out the divisor.
1240 LET J=J-1                             Resets J to try the same factor.
1250 RETURN
1260 PRINT
1270 PRINT
1280 PRINT "TO TEST A NEW NUMBER, ENTER 1  Presents option to rerun program.
1290 INPUT A
1300 IF A=1 THEN GO TO 1010
1310 STOP
```

Note: To run the program with 8K ROM, line 1130 must be changed to something like:
1130 IF INT(Q)=Q THEN GO SUB 1210

4K ROM
1K RAM



Defuse

Raymond Fowkes

You are the Chief of Security in a major government building. You have just received a telephone message from a terrorist group claiming that they have planted a bomb somewhere in the building. Fortunately, you have the most sophisticated electronic detection equipment available. Your detector gives off a signal that gets stronger as you get closer to the bomb. However, the building is large—100 stories high, 100 rooms long, and 100 rooms wide—and you are pressing a time limit of 200 seconds to find the bomb and defuse it.

Starting at the bottom (0, 0, 0), you enter the coordinates of each room you want to test (length, width, height) in response to the reading from your detector.

This program was adapted to the ZX80 and to fit into 1K of RAM from *More Basic Computer Games* published by Creative Computing.

Raymond Fowkes, P.O. Box 336, Coalinga, CA 93210.

Defuse Program Listing

```
 10 RANDOMISE
 20 LET A=RND(99)
 30 LET B=RND(99)
 40 LET C=RND(99)
 50 LET D=0
 60 LET E=0
 70 LET F=0
 80 LET S=0
 90 CLS
100 PRINT "#L####W####H####SECONDS##SIGNAL"
110 PRINT
120 LET L=PEEK(16421) (current line on screen)
130 IF L<22 THEN INPUT D
140 IF D<10 THEN PRINT "#";
150 PRINT D;
160 IF L<22 THEN INPUT E
170 IF E<10 THEN PRINT "#";
180 PRINT "##";E;
190 IF L<22 THEN INPUT F
200 IF F<10 THEN PRINT "#";
210 PRINT "##";F,S,10000-(ABS(A-D)+ABS(B-E)
    +ABS(C-F))*29
220 IF A=D AND B=E AND C=F THEN GO TO 270
230 IF S=200 THEN GO TO 310
240 IF L<4 THEN GO TO 90 (prevent screen overflow)
250 LET S=S+10
260 GO TO 110
270 CLS
280 PRINT
290 PRINT "BOMB DEFUSED AT ";S;" SECONDS"
300 GO TO 360
310 CLS
320 PRINT
330 PRINT "BOOOMMM THE BUILDING BLEW UP"
340 PRINT
350 PRINT "THE BOMB WAS AT ";A;",";B;",";C
360 PRINT
370 PRINT "PLAY AGAIN?"
380 INPUT A$
390 IF CODE(A$)=62 THEN RUN
```

Sample Run

L	W	H	Seconds	Signal
0	0	0	0	6636
40	0	0	10	6636
20	40	0	20	8376
20	60	0	30	8434
20	52	0	40	8666
20	56	0	50	8550
20	50	0	60	8666
20	51	16	70	9159
20	51	30	80	9565
20	51	46	90	9971
20	51	45		

```
-------------------------
BOMB DEFUSED AT 100 SECONDS
PLAY AGAIN?
"N"
```

Worth A Fortune

Past issues of Creative Computing. What are they worth today? It varies. To a collector, Vol. 1, No. 1 is worth $7 or $8. To a scrap dealer, less than two cents.

But we're not selling old back issues. We're all out.

On the other hand, you know that much of the content of Creative Computing is timeless. The Depth Charge program in Vol. 1, No. 1 is just as challenging today as the day it was written. Walter Koetke's series of five articles on using computers in the classrom are as valid today as the day they first ppeared in print. And scores of people have written about obtaining reprints of Don Piele's classic problem-solving series.

Our Mistake

In our early growth years when we had 5,000 and then 10,000 subscribers we couldn't imagine we would ever need more than 1000 extra copies for back issue sales. That's about what we printed extra. However, by the time we were going into Volume 3, we found our stocks of Volume 1 issues virtually depleted.

Our Solution

So we selected the best material from Volume 1, edited it, put it together in book form and sold it for $8.95, about the same

as the six individual issues. Nine months later, we did the same with Volume 2. Then a year and a half later we did it again with Volume 3.

Most other magazines in a high technology field like small computers find their contents are quickly out of date. However, because we've concentrated on applications and software, our content retains its value for a much longer time. Our subscribers know this and retain their copies of Creative Computing long after they've disposed of the more hardware-oriented magazines.

Now you can obtain the best material from the first three years of Creative Computing in book form and the next three years (minus four issues) in the original magazine form.

Our Offer

We have a unique special offer, so pay close attention to this paragraph. (Computer types ought to be able to understand this). If you order any one item below, you pay the full price. If you order any two items, take a 5% discount from the total; any three, take a 10% discount; any four, take a 15% discount, any five, take a 20% discount, and on all six take a whopping 25% discount from the total price.

Best of Creative Computing-Vol 1	$8.95
Best of Creative Computing-Vol 2	8.95
Best of Creative Computing-Vol 3	8.95
Volume 4 (Four issues)	6.00
Volume 5 (Ten issues)	15.00
Volume 6 (Twelve issues)	18.00

Less discount (5% for two items, 10% for three, 15% for four, 20% for five, 25% for all six) Shipping ($2.00 USA, $5.00 foreign)

We guarantee you'll never find a better value in computer applications reading matter. On average you're getting 128 pages of solid information for each $1.00. If you're not completely satisfied after you've read them, send the books or magazines back to us and we'll refund your full purchase price plus the return postage.

creative computing

Morris Plains, NJ 07950
Toll-free **800-631-8112**
(In NJ 201-540-0445)

ARTICLES AND COMMENTARY

- **Editorials**
  Birth of a Magazine — Ahl
  A Computer in the Classroom?
  Is Breaking Into A Timesharing System A Crime? — Tagg
  Where Are We Going? — Ahl
- **Computers in Education**
  What's Wrong With the Little Red Schoolhouse? — Ahl
  How to Cope With Your Computer
  Recent Trends in Mathem
  Curriculum Res
  CITALA: C
  How,
  EXPER SIP
  Monty Pyt
  IFIP Confer
- **Transporta**
  The Parable
  Technical Tr
  CONDUIT Do
  Statewide Po
  Expected Be
- **Hard Core C#**
  PLATO IV Syst
  TICCIT System
  PLANIT The Po
- **Careers**
  A Computer Car
  Career Educator
  Key to Your Futu
  Profile of an Indu
- **Applications**
  Computers and the
  Computer Simulatic
  Weather Forecastin
  Relativity for Computers: Alt Arithmetic
  Mr. Spock's 7th Sense — Kibler
- **Programming and Languages**
  Structured Programming — Hoogendyk
  On Computer Languages — Ahl
  Toward A Human Computer Language — Cannara
  Learning About Smalltalk — Goldeen
  Eclectic Programming Languages
  A New Approach to Testing —
- **Computer Impact on Society**
  The Computer Threat to Society — Ahl
  Digital Calculators — Then and Now
  The Computer Threat to Society? —
  Putting Teeth Into Privacy
  Legislation — Hastings
  Indust aders at Privacy
  : — Hastings
  ing in the Space Age —
  ver Looks at Data
  Fritze
  blic Attitudes Toward
  — Ahl
  onference
  vacy Should You Have —
  Ex-Social Security
  r — Campbell
  mputers — Malcolm
  igement Information
  Criminal Justice
  elman
  to the Computer
  :
  nal Computer
  gs
  Snyder

AND THINGS

  ... aushnell — Todd
  Playing PONG to Win — Ahl
  Your Own Computer? — Ahl
  Introducing Computer Recreations
  Corp. — Todd, Guthrey
  Creative Computing Compendium
  Flying Buffalo — Loomis
  Compleat Computer Catalogue
  National Computers in Education Conference?
  ... Exchange —

ARTICLES AND COMMENTARY

- **Technology — Present and Future**
  The Future of Computer Technology - Computing Power to the People
  Videodiscs — The Ultimate Computer Input Device? - *Bork*
  Round and Round They Go
  The $2.98 Computer Library - Personal Computers
  Russian Computing - *Ahl*
  Desk Calculator from Chi
  Microprocessors & Micro
  The State of the Art - C
- **Languages and Program**
  The Reactive Engine Pa
  About Computing - *Chi*
  David vs. 12 Goliaths -
  Sixth Chess Champior
  Beating the Game - *Tr*
  Simulated Strategies - *Reisman*
  Beyond BASIC - Sali
  The Computer Glas
  Teaching with AP
  Creative Chess - *Ko*
  SNOBOL - *Touretz*
  A Smalltalk Anthology
- **Artificial and Extra**
  Non-Human Intell
  An Esoteric Ethic:
  The Thinking Cor
  Primer on Artific
  Can Computers
  An Ear on the Ur
  Communicator
  The Cosmic Su...
- **Literacy, Philosophy, Opinion**
  What is Computer Literacy - *Moursund*
  Computer Literacy Quiz - *Moursund*
  A Fable - *Spero*
  Let Us First Make It - *Taylor*
  Some Thoughts - *Lees*
  Information Anyone? - *Griffith*
  The Government Dinosaur - *Winn*
  The Magic of EFTS - *Ahl*

- **Computers in Education**
  Instructional Computing in Schools - *Ahl*
  Should the Computer Teach the Student, or Vice-versa? - *Luehrmann*
  The Art of Education: Blueprint for a Renaissance - *Dwyer*
  Computing at the University of Texas
  Computers in Secondary Schools - 1975
  Compyouter Fair - *Thomas*
  The Madness known as
- **Every Person and the Computer**
  Amateur Computing - *Libes*
  ... Store? You Gotta Be *berts*
  ... Computer on
  ... of
  ... *Ahl*
  ... ner

... nbol - *Mueller*
... oetry - *Chisman*
... AcCauley
... er
... y

IS, AND PROGRAMS

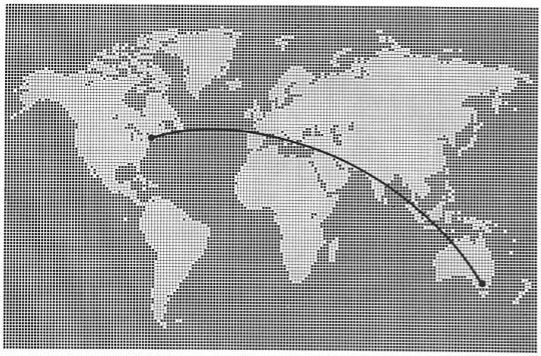
... ns
... uzzles - *Ahl*
... Recreations
... nto A Lesson - *Homer*
... r
... me - *Yarbrough*
... or Games - *Rogers*
... s

... d Geometry
The Mystic ... n - *Dickens*
Magic Squares on the Computer - *Piele*
Non-Usual Mathematics - *Reagan*
The World of Series - *Reagan*
Change For A Dollar - *Hess*
Sequences - *Jessen*
Progression Problems - *Reeves*

8K ROM
2K(1K) RAM

The Great Circle Route

Chuck Dawson



```
 10 REM "GREAT CIRCLE"
 20 PRINT "DEPARTURE POINT"
 30 GOSUB 400
 40 LET LAT1=LAT*PI/180
 50 GOSUB 450
 60 LET LONG1=LONG
100 PRINT "DESTINATION"
110 GOSUB 400
120 LET LAT2=LAT*PI/180
130 GOSUB 450
140 LET LONG2=LONG
150 CLS
160 LET D=(LONG1-LONG2)*PI/180
170 LET Z=SIN LAT1*SIN LAT2+(COS LAT1*COS LAT2*COS D)
180 LET DIST=INT (.5+(108000/PI*ACS Z))
190 PRINT AT 10,0;"DISTANCE#=#";DIST/10;"#NAUTICAL MILES"
200 PRINT "OR#";.115*DIST;"#STATUTE MILES"
210 STOP
300 SAVE "GREAT CIRCLE"
320 RUN
400 PRINT "LATITUDE"
410 GOSUB 500
420 LET LAT=DEG+MIN/60+SEC/3600
430 RETURN
450 PRINT "LONGITUDE"
460 GOSUB 500
470 LET LONG=DEG+MIN/60+SEC/3600
480 RETURN
500 PRINT "INPUT DEGREES#";
510 INPUT DEG
520 PRINT DEG
530 PRINT "INPUT MINUTES#";
540 INPUT MIN
550 PRINT MIN
560 PRINT "INPUT SECONDS#";
570 INPUT SEC
580 PRINT SEC
590 PRINT
600 RETURN
```

For those who have the 8K ROM. "The Great Circle Route" shows something of its possibilities. After entering the program on your computer, enter the latitude and longitude of any two points in the world, and the computer will calculate and display the distance between them. This is the "great circle route." The primary answer is in nautical miles, but this answer is converted to statute miles by line 200. Line 200 can be used to convert the answer to kilometers by changing the .115 multiplication factor to .185035. The program requires 2K RAM but it may be usable on 1K machines by using one letter names for the variables leaving out the seconds input portion, and omitting the conversion provision in line 200.

When you enter a longitude east of Greenwich or a latitude south of the equator, use negative numbers.

After years of looking up trig functions in tables, I am amazed by the speed at which this program runs.

Chuck Dawson, 6520 Victoria, Fort Worth, TX 76118.

The ZX80/1 As Fortune Cookie

Craig Breighner

易
經

In addition to reputed oracular powers, the *I Ching* has proven to be a source of fascination for mathematicians and computer scientists. This ancient Chinese system of divination comprises one of the earliest known examples of a binary counting scheme.

The *I Ching* [*Book of Changes*] uses solid and broken lines to construct curious little six-line figures or "hexagrams" like the one shown below:

Since there are only two possible values for each of the six lines (solid/broken) it is not surprising that there are 64, or 2^6 hexagrams.

Associated with each hexagram is a special interpretation based on the interrelationship of solid and broken lines in the figure. This interpretation is believed to hold a particular relevance for the person for whom the hexagram has been constructed.

There are several ways to construct an I Ching hexagram. Some methods use either milfoil or yarrow stalks. The stalks are hard to come by, so bamboo is often substituted. But the complexity of some of these methods—it is said that they can only be learned from a master—would seem to defy any kind of straightforward algorithmic definition (at least any that would fit into 1K of RAM).

However, there are other methods. One is to use coins. This method has great popularity even in the Orient. A relatively simple version follows:

1) Toss three coins simultaneously.

2) Assign "point values" to the coins—2 for heads; 3 for tails.

3) Add up the total points for all three coins.

4) If the total equals 6 or 8 draw a broken line (– –).
If the total equals 7 or 9 draw a solid line (——).

5) Repeat this procedure until six lines have been drawn, *bottom* to *top*.

Now, this method can be readily simulated on the ZX80. The accompanying Basic program can be lots of fun at parties. And who knows? It may even hold a deeper significance for those who consider themselves to be particularly "in tune" with their machines.

The program uses two nested FOR... NEXT... loops. The outer loop controls the number of lines in the hexagrams (6); the inner loop the number of coins (3) to be tossed.

Unfortunately, the program does not tell the user what his or her hexagram *means* (What did you expect in 1K, anyway?). But an excellent source of in-depth interpretations for each hexagram is *I Ching: Book of Changes*, Causeway Books, New York. This is an inexpensive reprint of James Legge's 1882 translation. It is still highly regarded both for its accuracy and its completeness. Because of Legge's notoriety as an orientalist, this translation should be available at most large municipal and university libraries.

```
 100 DIM I(6)
 110 PRINT "DOES YOUR FUTURE LIE
#IN HARMONY WITH THE";
 120 Print "#ELECTRONIC PULSE OF
#THEUNIVERSE?"
 130 PRINT
 140 PRINT "CONSULT THE ORACLE O
F THE","I CHING: PRESS";
 150 PRINT "#NEWLINE TO CREATEAN
#I CHING HEXAGRAM."
 160 INPUT A$
 170 FOR J=1 TO 6
 180 LET K=Ø
 190 FOR L=1 TO 3
 210 LET K=K+INT (RND*2)+1
 220 NEXT L
 230 LET I(7-J)=K
 240 NEXT J
 250 CLS
 260 PRINT "YOUR HEXAGRAM:"
 270 PRINT
 280 FOR J=1 TO 6
 290 LET K=I(J)
 300 IF K=4 OR K=6 OR K=8 THEN G
OTO 330
 310 PRINT "▬▬▬"
 320 GOTO 34Ø
 330 PRINT "▬ ▬"
 340 NEXT J
```

Notes:

310: Use the graphic on the 6 key 5 times.

330: Use the graphic on the 6 key 2 times, space, 2 times.

This program will work on 4K ROM with the following changes:

210 LET K=K+RND(2)+1

Use the graphic on the W key in lines 310 and 330. ■

Craig Breighner, Present Time Systems, 1250 Dagmar Ave., Pittsburgh, PA 15216. Program edited to 8K ROM by David Grosjean.

4K ROM
2K(1K) RAM

Hangman

Raymond Fowkes

The old game of *Hangman* can also be played on your ZX80. The program listed below preserves not only the battle of wits between the two players, but also draws on the graphics capabilities of the ZX80 to draw the figure for you. The program with full prompts on the screen requires over 1K, but it can be played on 1K by making the changes listed at the end of the program.

After typing in your program, press RUN and NEWLINE. The first player then types in a secret word of up to 17 characters. After clearing the screen, the computer will indicate by dashes how many letters the word has. The other player then attempts to guess the word by typing in letters most likely to be in the word. A right letter is placed in its correct position(s). A wrong or repeated letter is printed in the center of the screen and a body part is added to the figure on the gallows. The game ends when the figure is completed or when the secret word is filled in. ■

Raymond Fowkes, P.O. Box 336, Coalinga, CA 93210.

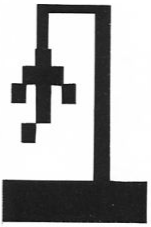
Sample Runs

1. HANGMAN
INPUT SECRET WORD.
"SOFTWARE"

2.

SECRET WORD: ------

3.

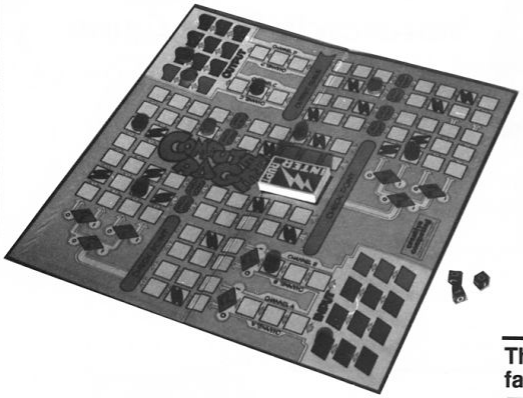


SECRET WORD: SOFTWARE
I N G C B M H P D (Player's guesses)
YOU GUESSED IT.
THE WORD WAS: SOFTWARE
PLAY AGAIN?
"YES"

Hangman Program Listing

```
 10 CLS
 20 PRINT ,"##HANGMAN"
 30 PRINT
 40 PRINT
 50 PRINT "INPUT SECRET WORD."
 60 INPUT A$
 70 CLS
 80 PRINT "###TTTTTT"  (6 shift T)
 90 PRINT "###Q####A"  (shift Q; shift A)
100 PRINT "###Q####A"  (shift Q; shift A)
110 FOR A=1 TO 7
120 PRINT "########A"  (shift A)
130 NEXT A
140 PRINT "AAAAAAAAAAAA"  (12 shift A)
150 PRINT "AAAAAAAAAAAA"  (12 shift A)
160 PRINT
170 PRINT
180 PRINT "SECRET WORD:##";
190 LET B$=A$
200 DIM A(17)
210 FOR E=0 TO 17
220 IF B$="" THEN GO TO 310
230 LET A(E)=CODE(B$)
240 PRINT "-";
250 LET B$=TL$(B$)
260 NEXT E
310 PRINT
320 PRINT
330 LET F=0
340 LET B$="#I2AGO4CLNUY"
350 INPUT C$
360 LET B=0
370 FOR A=0 TO E-1
380 IF CODE(C$)=A(A) THEN GO SUB
    500
390 IF E=F THEN GO TO 550
400 NEXT A
420 IF B THEN GO TO 350
430 PRINT CHR$(CODE(C));"#";
440 LET C=CODE(B$)
450 IF C>15 THEN LET C=C-10
460 POKE 34+PEEK(16396)+PEEK(16397)
    *256+C,9
470 LET B$=TL$(B$)
480 IF B$="" THEN GO TO 590
490 GO TO 350
500 POKE 143+PEEK(16396)+PEEK(16397)
    *256+A,CODE(C$)
510 LET A(A)=-1
520 LET F=F+1
530 LET B=1
540 RETURN
550 PRINT
560 PRINT
570 PRINT "YOU GUESSED IT."
580 GO TO 620
590 PRINT
600 PRINT
610 PRINT "YOU LOSE."
620 PRINT
630 PRINT "THE WORD WAS:#";A$
650 PRINT
660 PRINT "PLAY AGAIN?"
670 INPUT A$
680 IF CODE(A$)=62 THEN RUN
```

This program may be converted for 1K by doing the following:
Delete lines 20-40 and 550-610
Change line 620 to: 620 CLS



High Roller

Three binary dice add up to fast fun and easy winnings

Binary dice? That sounds strange. What's the point?

Each binary die has six sides but instead of one to six spots, three sides have the numeral one and three have a zero. When rolled, the three dice, red, blue and green, produce a 3-bit binary number.

The binary number can be easily converted to a decimal numbr. A binary 101 equals a decimal 5. After using these dice a few times, these conversions are quickly done even by 7 year olds.

Designed for Understanding

Binary dice are just one of many unique elements of the *Computer Rage* board game. The whole game is designed to help players easily understand the complexities of a large multiprocessing computer system while having great fun playing.

Imagine you're using a large computer along with many other users. It's Thursday and payroll checks have to be run. They have priority over your job. When this happens in the game you lose a turn. But then a vice president wants the results from the program you're working on—take another turn. Oh, oh, in your hurry, you make a program mistake. Too bad—return to the last checkpoint.

Meanwhile one of your opponents, a fellow user of the computer, has heard from the president that one of his three programs has top priority; it advances to the output queue. But wait, on your next move you land on an "interrupt" and find that a brownout has just occurred, the computer has crashed, and all the programs of all players must return to the last checkpoint.

The binary dice return to your opponent. He rolls 011, a four, and lands on a decision point. Rolling one die he gets a 1 which means he takes an 8-step flow intead of a 16-step one.

One it goes until one of the two to four players gets all of his programs to the output printer and wins.

Sets of three binary dice used in Computer Rage are available separately.

Sturdy Components

The game comes with a colorful, big 19" x 19" playing board, 38 interrupt cards, 12 miniature disk pack playing pieces (3 for each player) and 3 binary dice. A supplement to the rules describes the way in which *Computer Rage* parallels a multiprocessing system.

Computer Rage is designed for players from 7 to 14 years old but obviously can be played by adults as well. It is for two to four players. Many schools use the game along with a book such as *Be A Computer Literate* in computer literacy units for Grades 3 to 8. It's also an excellent game to have available in open or alternative classrooms.

Discounts Available

In fact we feel so strongly that *Computer Rage* should be in every school that we're offering a special discount to schools and to people who buy a game for a school.

The price of one game is $8.95 postpaid. Buy one game for your family and another for a school and the total price is just $14.00 postpaid (and $8.95 is tax deductible). Individuals or schools buying five or more games may take a whopping 50% discount—just $4.50 each. Customers outside of the U.S. must add $2.00 additional postage per game.

If you'd like an extra set of three binary dice for home or classroom, they are available for just $1.25 per set or five sets for $5.00.

Order today at no risk. If you're not completely satisfied, return the game or dice for a full refund. To order, send your check or charge card number to the address below. Visa, MasterCard, and American Express orders may be called in toll-free to **800-631-8112** (in NJ 201-540-0445). School purchase orders should add an additional $1.00 billing fee.

Don't put it off. Order this entertaining and educational game today.

creative computing

Morris Plains, NJ 07950
Toll-free **800-631-8112**
(In NJ 201-540-0445)

Motorcycle Race Game

Richard Van Workum

4K ROM
1K RAM

Test your skill and luck on a ZX80 motorcycle! In this game you compete with another rider on a 32 mile track on which you meet various obstacles and road conditions. The key to winning the race is meeting the obstacles with just the right speed. The faster you try to go, the less chance you have of overcoming the obstacle. If you go too fast, you will go down; if you choose the right speed, you will advance after the obstacle according to the speed you choose. Of course, you cannot play it safe by sticking to the lower speeds because your opponent will beat you to the finish line. You must take the risks to win. No trophies are offered, but you can still be the winner.

Type in the program and run it. The computer will print out a symbolic track and show the relative positions of the cycles as the challenges of the track are met. The first player will be asked how fast he wants to take obstacle #1 then presses a number from 1 to 9 and NEWLINE. The second player is asked how fast he wants to take obstacle #1. He also enters his choice. After NEWLINE is pressed, the computer will show the positions of the motorcycles according to the players' choices. It will print how far each cycle advances and in parentheses the total miles travelled. If the cycle goes too fast, it will be down. A new obstacle is presented, and each player is asked how fast he wants to go to meet it. The play continues until one cycle (or both) completes the 32 mile course and the winner declared.

Richard Van Workum, 920 Leslie Ln., Hanford, CA 93230.

Sample Run

```
GRAY CYCLE
▀
BLACK CYCLE
▀
FALLEN RIDER
HOW FAST?

GRAY CYCLE INPUT 1 TO 9

LS
```

Figure 1. First player's turn.

```
GRAY CYCLE
 ▀
BLACK CYCLE
▀
FALLEN RIDER
HOW FAST?

GRAY CYCLE INPUT 1 TO 9

BLACK CYCLE INPUT 1 TO 9

LS
```

Figure 2. Second player's turn.

```
GRAY CYCLE
  ▀
BLACK CYCLE
 ▀
GRAY CYCLE ADVANCES 4    (5 MI.)
BLACK CYCLE ADVANCES 3   (4 MI.)

LOOSE GRAVEL
HOW FAST?

GRAY CYCLE INPUT 1 TO 9

LS
```

Figure 3. Race status after first turns.

```
  2 LET G=1
  3 LET B=1
 12 LET A$="BLACK CYCLE"
 14 LET B$="GRAY CYCLE"
 15 GO SUB 815
100 LET D=RND(6)
104 IF D=1 THEN PRINT "WATER AN
D MUD"
106 IF D=2 THEN PRINT "DEEP HOL
E"
108 IF D=3 THEN PRINT "SHARP TU
RN"
110 IF D=4 THEN PRINT "BUMPY TR
ACK"
112 IF D=5 THEN PRINT "FALLEN R
IDER"
114 IF D=6 THEN PRINT "LOOSE GR
AVEL"
120 PRINT "HOW FAST?"
135 PRINT
140 LET D=RND(10)
152 PRINT B$;" INPUT 1# TO 9"
155 INPUT K
156 PRINT
167 PRINT A$;" INPUT 1# TO 9"
170 INPUT U
180 IF NOT K>D THEN LET G=G+K
181 IF NOT U>D THEN LET B=B+U
182 IF G>32 THEN LET G=32
185 IF B>32 THEN LET B=32
200 GO SUB 815
210 IF NOT K>D THEN PRINT B$;"#
ADVANCES#";K,"(";G;"#MI.)"
220 IF K>D THEN PRINT B$;"#IS D
OWN","(";G;"#MI.)"
230 IF NOT U>D THEN PRINT A$;"#
ADVANCES#";U,"(";B;"#MI.)"
240 IF U>D THEN PRINT A$;"#IS D
OWN","(";B;"#MI.)"
242 PRINT
245 IF G<32 AND B<32 THEN GO TO
90
260 GO SUB 815
280 IF G=B THEN PRINT ,"TIE"
285 IF G>B THEN PRINT ,B$;"#WON
"
290 IF B>G THEN PRINT ,A$;"#WON
"
310 STOP
815 CLS
845 PRINT B$
850 GO SUB 880
860 POKE PEEK(16396)+PEEK(16397
)*256+11+G,139
870 PRINT A$
880 FOR S=1 TO 32
885 PRINT "W";  (shift)
890 NEXT S
895 POKE PEEK(16396)+PEEK(16397
)*256+57+B,128
900 PRINT
901 PRINT
905 RETURN
```

Creative Computing-- Albert Einstein in black on a red denim-look shirt with red neckband and cuffs.

Creative's own outrageous **Bionic Toad** in dark blue on a light blue shirt for kids and adults.

Plotter display of **Pi to 625 Places** in dark brown on a tan shirt.

I'd rather be playing spacewar-- black with white spaceships and lettering.

Give your tie a rest!

All T-shirts are available in adult sizes S,M,L,XL. Bionic Toad, Program Bug and Spacewar also available in children's sizes S(6-8), M(10-12) and L(14-16). Made in USA. $6.00 each plus 75¢ shipping.

Specify design and size and send payment to Creative Computing, 39 E. Hanover Ave., Morris Plains, NJ 07950. Orders for two or more shirts may be charged to Visa, MasterCard or American Express. Save time and call toll-free **800-631-8112** (in NJ 201-540-0445).

Crash Cursor and Sync from the comic strip in SYNC magazine emblazoned in white on this black shirt.

Computer Bum-- black design by cartoonist Monte Wolverton on gray denim-look shirt with black neckband and cuffs.

The **Program Bug** that terrorized Cybernia in Katie and the Computer is back on this beige t-shirt with purple design. You can share the little monster with your favorite kid.

Roll down the block with this little black **Robot Rabbit** (on a bright orange t-shirt) on your back and you can intimidate every carrot, radish or cuke in your way.

Hints & Tips For The ZX81

Martin Wren-Hilton

Hints & Tips for the ZX81 by Andrew Hewson is excellent for the average ZX81 user who has little, or no knowledge of Z80 Machine Code programming, but would like to learn. There are sections on saving space in programs, understanding the Display File, converting ZX80 programs for use on the ZX81, how to let two or more programs access one lot of data and finally, 26 pages on Machine Code programming.

The book begins by showing the reader how to save a considerable amount of RAM by carefully designing programs. With so much memory used for system variables, this chapter is essential reading for ZX81 users with only 1K RAM.

The second chapter deals with the Display File—showing how TAB, AT, PLOT and UNPLOT work. This chapter also details how the Display File can be used as memory.

The next section discusses the similarities and differences between the ZX80 and ZX81 and how to convert programs between the two machines. Two conversion charts are given—one for converting character codes from ZX80 to ZX81, the other compares the ZX80 system variables with their ZX81 equivalents.

Chapter four demonstrates how two or more programs can share one lot of data by transferring the data into RAMTOP before loading the next program. This technique also allows for data to be 'NEW-proofed' in a way in which the data can be subsequently retrieved and processed by another program.

The final chapter explains to the complete novice how to write, load, edit, save, and debug Z80 Machine Language programs. The reader is first introduced to the concept of bits, bytes, addresses and hexadecimal numbering.

The author goes on to describe the Z80A registers, their relative uses, and the Z80A instruction set. Obviously, a book of this size cannot provide a full examination of the machine code of the Z80A microprocessor, but it does introduce the beginner to many basic concepts.

Hints & Tips for the ZX81 ends with thirteen programs for the basic, unexpanded ZX81. These programs include a useful Hex Loader/Printer, a Line Renumber program which neatens programs but does not include GOTO or GOSUB renumber, five games, a machine code clock program, a statistics program which generates the mean, standard deviation, intercept and slope of a given set of data.

Hints & Tips is clearly typeset, but is only bound with a couple of staples. It is a very good value for the money and will introduce many to the complex world of machine code programming.

Hints & Tips for the ZX81 by Andrew Hewson from: Hewson Consultants, 7 Grahame Close, Blewbury, Oxfordshire, OX11 9QE England. 76 pp. softbound; £4.95. ■

Martin Wren-Hilton, 4 Little Poulton Lane, Poulton-le-Fylde, Blackpool, FY6 7ET, England.

The "QS Sound Board" For The ZX80/81

Play tunes in three-part harmony on your ZX80 or ZX81! Based on the extremely versatile AY-3-8910 sound generator chip, the QS Sound Board features complete software control of the frequency and amplitude of three independent output channels as well as an envelope shaper and noise channel.

The QS Sound Board can produce fairly accurate scales over a 5 octave range, from C at 32.7Hz to B at 989Hz with a minimal error of +/-0.5%. Because of the limitations of the power supply, no amplifier or speaker has been fitted to the QS Sound Board. Instead, the three channels have been mixed together and taken to a 3.5mm jack socket via a preset volume control, therefore an external amplifier and speaker are needed. The Radio Shack 277-1008 is recommended.

If you wish to use more than the onboard memory with the QS Sound Board, you will need the QS Motherboard which allows the 16K RAM pack to be used in conjunction with the QS Sound Board and one other board.

The QS Sound Board also features two 8 bit input/output ports taken to a 16 pin i/c socket for easy connection to external control functions via ribbon cable.

The prices for the above products are:

QS Sound Board £28.00
QS Motherboard £13.00
QS Sound Board & Motherboard £38.00

Quicksilva are at 95 Upper Brownhill Road, Maybush, Southampton, Hants., England.

ZX81 Hardware Review (UK)

Producer	1	2	3	4	5	6	7	8
48K RAM pack	.	.	.	.	.	148.35	.	.
16K RAM pack	.	36.99	.	.	54.05	.	.	49.95
8K RAM pack	.	33.95	.	.	.	.	.	.
4K RAM pack	18.00	19.95	.	.	.	.	.	.
Motherboard	13.00	.	.	.	.	.	16.25	.
Sound board	28.00	.	.	.	.	.	U/D	.
Character Generator	25.00	.	34.50	.	.	.	.	.
Color board	.	U/D	U/D	.	.	.	.	.
Full-size Keyboard	.	24.95	.	.	.	.	27.74	.
Disc drive	.	.	.	U/D	.	.	.	.
Printer	.	.	.	U/D	.	.	.	49.95

All Prices in £ Sterling.
U/D - Under Development

1) Quicksilva, 95 Upper Brownhill Road, Maybush, Southampton, Hants.
2) dK'tronics, 23 Sussex Road, Gorleston-on-Sea, Gt.Yarmouth, Norfolk.
3) Haven Hardware, 4 Asby Road, Asby, Workington, Cumbria.
4) Macronics, 26 Spiers Close, Knowle, Solihull, W.Midlands, B93 9ES.
5) Audio Computers, 87 Bournemouth Park Road, Southend-on-Sea, Essex.
6) Memotech, 103 Walton Street, Oxford.
7) Redditch Electronics, 21 Ferney Hill Avenue, Redditch, B97 4RU.
8) Sinclair Research Ltd., 6 Kings Parade, Cambridge, CB2 1SN.

try this

This column will feature short programs to show off your ZX80, impress your family and friends, and tickle your imagination when *SYNC* arrives at your place. We invite your contributions. Address them to *SYNC*. 39 E. Hanover Ave., Morris Plains, NJ 07950.

Enter:

```
10 PRINT USR(722)
20 LIST USR(722)
```

Press RUN and NEWLINE.

Hit RUBOUT and continue until only the cursor is left on the screen. Note the effects

While the cursor is visible, type in your name.

Our thanks to:

David Plumb
34 Hillside Crescent
Enfield, Middx.
London EN2 OHR
England

We introduce in this issue the "Kitchen SYNC" by Alan Groupe, Michael Tardiff, and Ivan Zatkovich. This is not a column on cooking with the ZX80/1 (although conceivably that could be covered). Rather, the wide ranging interests of this trio are suggested.

Furthermore, they are open to suggestion for topics you want to hear more about. We will pass along all requests.

Kitchen SYNC

Alan Groupe, Michael Tardiff, and Ivan Zatkovich

Expression Evaluators at Work

The three of us work for Digital Equipment Corporation, and, in the course of our work, we have unlimited access to a great number of large computers. Yet we each recently bought ZX80s. Why?

We were warned that the ZX80 was a relatively tiny computer, almost a toy, with limited capabilities and a "very small" amount of memory. But it was the restrictions and the limitations of the machine that interested us. The computers we use every day have millions of bytes of memory; operating systems occupy tens of thousands of bytes of memory; programs have all the RAM they need and more. There is comparatively little need to "economize" in writing software. When such a need arises, we talk of shrinking a 20K program to fit into 16K. We decided that it would be fun to see just how far a "little" machine could be pushed.

Since then, we have become impressed with some of the features this compact machine does offer, and we would like to share some of our observationss and discoveries with you.

One of the first unusual (to us) properties of the ZX80 version of Basic that we "discovered" is that anywhere you need to enter a number, you can enter an equation instead. Or, to put it more impressively, if less comprehensibly, the expression evaluator is called at each instance of a value-required context.

What does that mean, and why is it good?

If a computer language is to accept and solve equations, which we will call "expressions," it must have the ability to take an expression as input and return a numeric value as output. This portion of the language is called the "expression evaluator."

The expression evaluator first gets values for the variables in an expression and then performs the indicated arithmetic operations to end up with a single value. For example, enter the following commands into your ZX80 in the immediate mode (that is, without typing line numbers first).

```
LET A=5
LET B=3
LET X=A+B
PRINT X
```

You should have seen an "8" at the top of your screen. When you entered the third statement, the expression evaluator looked in memory and found the value of A (which was 5) and the value of B (set to 3 in the second LET statement), then added them together and filed the result under X.

While normally the expression evaluator is only used to handle arithmetic statements, like LET; on the Sinclair machine it is used anywhere a number can be entered. For example, in a GO TO statement, you could insert an expression in place of the statement number of the GO TO. Instead of:

```
GO TO 40
```

you could write

```
GO TO X+10
```

If X equals 30, the expression evaluator would first search out the value of X, add 30 and 10, and then "GO TO" the result: statement number 40.

In a machine like the ZX80, it is not much trouble for Basic to use the expression evaluator often, but it certainly can be very handy for us in writing programs.

For an example, enter and run the following small program:

```
10 INPUT LAST
20 PRINT LAST
30 GO TO 10
```

When you are prompted for input, enter the number 3. The number 3 will appear on the screen. Enter an 11 and an 11 appears. On lesser machines (like the TRS-80, etc.) this is all the program will do. But on the Sinclair, you have just written a simple calculator! Enter 3+9 and 12 appears on the screen. 4*7 gives you a 28. In fact, you can even use the previous answer in an expression (assuming you have typed in at least one expression previously). Enter LAST-5 and the Sinclair responds with 23 (assuming you been entering all the examples).

When you entered 3, the expression evaluator was used. It evaluated the expression and returned the result (3), which was stored in the variable LAST. When you entered 4*7, the expression evaluator evaluated the expression to 28 which was stored in LAST. And when you entered LAST-5, the expression evaluator recalled the value for LAST (28), subtracted 5 from it, and returned the result (23) which was stored in LAST.

Another posible use of this technique is in the following rather crude telephone directory:

```
 10 PRINT "ALAN GROUPE"
 20 PRINT "ALANS ADDRESS"
 30 PRINT "ALANS PHONE"
 40 STOP
 50 PRINT "MICHAEL TARDIFF"
 60 PRINT "MICHAELS ADDRESS"
 70 PRINT "MICHAELS PHONE"
 80 STOP
 90 PRINT "IVAN ZATKOVICH"
100 PRINT "IZZIES BAR AND GRILL"
110 PRINT "IVANS PHONE"
120 STOP
```

Running this program by entering "RUN" is of little value, since it will always print only the first person's information. So you must run the program with the GO TO command rather than the RUN command. If you want Alan's information, you enter GO TO 10. If you want Ivan's, you enter GO TO 90.

This is all well and good, except that remembering the numbers is probably harder than remembering the addresses and phone numbers. However, after typing in the program, you can enter the following commands in immediate mode (that is, without typing line numbers first):

```
LET GROUPE=10
LET TARDIFF=50
LET ZATKOVICH=90
```

Now, if you want information on Ivan Zatkovich, you only need to enter GO TO ZATKOVICH. Again the expression evaluator is used. It determines that the variable ZATKOVICH has the value of 90 and then GOes TO statement 90. When you save a program on tape, the values of the variables are also saved; so when you load your telephone directory again you will not have to re-enter the LET statements. A word of warning though—the RUN command clears the values of all the variables. If you accidentally enter RUN, you will have to re-enter all the LET statements or LOAD the tape again.

The expression evaluator will also work with strings as well as with numbers, although we do not see a use for this feature at the moment. Run the following program:

```
10 INPUT A$
20 PRINT A$,CODE(A$)
30 GO TO 10
```

You will notice that the cursor appears on the screen between a pair of quotation marks(""). Enter the letter A. On the screen will appear the letter A and the number 38, which is the ZX80's internal numeric code for representing the character A. Now, using the RUBOUT and arrow keys, rubout the two quotation marks. You will notice that the familiar syntax error symbol appears. This is because the INPUT statement is looking for a string, and strings are delimited by quotation marks. If you enter "A" (including the quotation marks) you will see that it is accepted, because "A" is a valid string no matter whether you typed the quotation marks or whether the machine did it for you.

Certain internal functions of the ZX80 (those whose names end in $, such as CHR$), return strings as their outputs. As before, rubout the two quotation marks, but this time enter CHR$(38) (no quotation marks). Once again, the expression evaluator is used and evaluates the expression, returning the string "A" as its output. Since this is a valid string, it satisfies the INPUT statement and is accepted.

Now it is up to you to figure out a use for this feature of the ZX80. Send your discoveries to *SYNC*.

resources

Software

- Exclusive distributors for the ZX80/1 technical documentation. ZX80/1 information, software, accessories, etc. Send for a free catalog to:
  The ZX-GROUP
  25 Shute Path
  Newton, MA 02159

- Free ZX80 game software.
  Just send your original 8K ROM software as listings or on tape and I will send you an equal number of mine.
  Ken Wogaman
  P.O. Box 125
  Somerville, OH 45064

- ZX80/MicroAce Users!
  1K—*Computerized Cowboy*, *Assassin*, *Energy Costs*, *Home Inventory*, *Phone Number Organizer*, and more!
  16K-*French Vocab.*, *Deluxe Computerized Cowboy*.
  Send SASE for more info. 1K—$5 (2 for $9); larger programs—$10 (2 for $17). Send to:
  Steven Karash
  ZX Software
  11 Holland Lane
  New Paltz, NY 12561

- *ZX80 Chess* developed specifically for the ZX80. 3 levels of play against the computer; graphics board displayed; fairly rapid response time. $19 from:
  ZX80 Chess (American)
  P. Joy
  130 Rush Green Road
  Essex, RM7 0QA
  England
  Ask for details on other software; SAE.

- Program Techniques: *Machine Language*, *Sorting*, others. Educ.: *Math (+-x/)*, *Hi Prec. Div.*, *Hex/Dec Conv.*, *Pound Dollar Conv.*, *Temp. Conv.*, *Graphing*, others. Games: *Submarine*, *Craps*, *Lunar Lndr*, others. $4.95 ea; any 3 $12; all 1K; check or money order to:
  The Italian Connection
  869 Levitt Pkwy
  Rockledge, FL 32955

- Fascinating *Life* Graphics program; 22x27 world; half second between displays. 4K ROM; 1K RAM. $1 for listing, instructions, and patterns to:
  Jon T. Passler
  344 Cabot St
  Beverly, MA 01915

- *ZXCHESS* and *Adventure 'A'* in 4K or 8K ROM; 16K RAM. Machine code. 6 levels of chess against the ZX80/1; algebraic notation; displays normal video for white; inverse, black. £10 UK; $25 US; incl. shipment to US.
  Adventure 'A' A search for your space ship to escape a hostile planet. First in a series. £7 UK: $20 US incl. shipment to US.
  Artic Computing
  396 James Reckitt Avenue
  Hull, North Humberside
  United Kingdom

Hardware

- Remote Switched Isolators.
  3 sockets each capable of 1KW load. Prices start at $79.95. For details:
  Electronic Specialists, Inc.
  171 S. Main St.
  Natick, MA 01760
  Phone: (617) 655-1532

- 8K/4K ROM switch kit.
  Allows switch selection of ROMs. $24.95. Cassette load AGC/audio monitor; keyboard beeper option; add $10.00 to basic switch kit price.
  Marex Electronics
  2805 Abbeyville Rd.
  Valley City, OH 44280

- Overvoltage protector, Model OV-1.
  A plug-on unit that saves your ICs if regulator fails. Also has computer on/off switch & allows operation on 115 v. AC or 12 v. DC. $35 + $2 shipping.
  USA only.
  Quest Research Associates
  P.O. Box 3073
  San Jose, CA 95156-3073

- 16K RAM Expansion plus power supply, model MX-16, $89.95 + $3 shipping/handling. Check or money order to:
  INSIGHT
  1889 Lewis Dr.
  Niles, MI 49120
  (or call 616-684-7868 for COD; no extra charge for COD)

- 48K Memory Extension for ZX81.
  Allows ZX81 to run 48K Basic programs including up to 16K of assembly code. ZX81 sits on custom built case containing the Memotech. £109 + 15% VAT kit form: £129 + VAT assembled. Send to:
  Memotech (Sales Dept.)
  103, Walton Street
  Oxford OX2 6EB
  England

- ZX80/1 User Port.
  A small board allowing 8 independent input channels and 8 output channels, directly controlled from Basic or machine code. Details on uses included in booklet. Complete kit—£11.50; booklet of applications—£.40; suitable loudspeaker—£.80. Add VAT and P&P.
  Technomatic
  17 Burnley Rd., NW 10
  01 452 1500
  England

- ZX Printer. £49.95 by mail order from:
  Sinclair Research Ltd
  6 Kings Parade
  Cambridge, CB2 1SN
  England

- The 55 Key keyboard from Schultz Systems (*SYNC* 1:3) is no longer available. Stocks are exhausted.

User Groups

- Bay Area ZX80 User Group
  2660 Las Aromas
  Oakland. CA 94611

- ZX80/1 User Group within the Nottingham Micro-Computer Club. Contact:
  G. E. Basford
  9 Holme Close
  The Pastures
  Woodborough
  Nottingham NG1 4 6 EX
  England

David Ahl, Founder and Publisher of Creative Computing

creative computing

"The beat covered by Creative Computing is one of the most important, explosive and fast-changing."—Alvin Toffler

You might think the term "creative computing" is a contradiction. How can something as precise and logical as electronic computing possibly be creative? We think it can be. Consider the way computers are being used to create special effects in movies—image generation, coloring and computer-driven cameras and props. Or an electronic "sketchpad" for your home computer that adds animation, coloring and shading at your direction. How about a computer simulation of an invasion of killer bees with you trying to find a way of keeping them under control?

Beyond Our Dreams

Computers are not creative per se. But the way in which they are used can be highly creative and imaginative. Five years ago when *Creative Computing* magazine first billed itself as "The number 1 magazine of computer applications and software," we had no idea how far that idea would take us. Today, these applications are becoming so broad, so all-encompassing that the computer field will soon include virtually everything!

In light of this generality, we take "application" to mean whatever can be done with computers, *ought* to be done with computers or *might* be done with computers. That is the meat of *Creative Computing*.

Alvin Toffler, author of *Future Shock* and *The Third Wave* says, "I read *Creative Computing* not only for information about how to make the most of my own equipment but to keep an eye on how the whole field is emerging.

Creative Computing, the company as well as the magazine, is uniquely lighthearted but also seriously interested in all aspects of computing. Ours is the magazine of software, graphics, games and simulations for beginners and relaxing professionals. We try to present the new and important ideas of the field in a way that a 14-year old or a Cobol programmer can understand them. Things like text editing, social simulations, control of household devices, animation and graphics, and communications networks.

Understandable Yet Challenging

As the premier magazine for beginners, it is our solemn responsibility to make what we publish comprehensible to the newcomer. That does not mean easy; our readers like to be challenged. It means providing the reader who has no preparation with every possible means to seize the subject matter and make it his own.

However, we don't want the experts in our audience to be bored. So we try to publish articles of interest to beginners and experts at the same time. Ideally, we would like every piece to have instructional or informative content—and some depth—even when communicated humorously or playfully. Thus, our favorite kind of piece is acessible to the beginner, theoretically non-trivial, interesting on more than one level, and perhaps even humorous.

David Gerrold of *Star Trek* fame says, "*Creative Computing* with its unpretentious, down-to-earth lucidity encourages the computer user to have fun. *Creative Computing* makes it possible for me to learn basic programming skills and use the computer better than any other source.

Hard-hitting Evaluations

At *Creative Computing* we obtain new computer systems, peripherals, and software as soon as they are announced. We put them through their paces in our Software Development Center and also in the environment for which they are intended—home, business, laboratory, or school.

Our evaluations are unbiased and accurate. We compared word processing printers and found two losers among highly promoted makes. Conversely, we found one computer had far more than its advertised capability. Of 16 educational packages, only seven offered solid learning value.

When we say unbiased reviews we mean it. More than once, our honesty has cost us an advertiser—temporarily. But we feel that our first obligation is to our readers and that editorial excellence and integrity are our highest goals.

Karl Zinn at the University of Michigan feels we are meeting these goals when he writes. "*Creative Computing* consistently provides value in articles, product reviews and systems comparisons . . . in a magazine that is fun to read."

Order Today

To order your subscription to *Creative Computing* send payment to the appropriate address below. Customers in the continental U.S. may call toll-free to charge a subscription to Visa, MasterCard or American Express.

		Canada and	
Term	*USA*	*Foreign Surface*	*Foreign Air*
1 year	$20	$29 or £12.50	$50 or £21
2 years	$37	$55 or £24.00	$97 or £41
3 years	$53	$80 or £34.50	$143 or £61

We guarantee your satisfaction or we will refund your entire subscription price.

Join over 80,000 subscribers like Ann Lewin, Director of the Capital Children's Museum who says, "I am very much impressed with *Creative Computing*. It is helping to demystify the computer. Its articles are helpful, humorous and humane. The world needs *Creative Computing*."

creative computing

P.O. Box 789-M
Morristown, NJ 07960
Toll-free **800-631-8112**
(In NJ 201-540-0445)

27 Andrew Close, Stoke Golding
Nuneaton CV13 6EL, England

SYNTAX ZX80®

A PUBLICATION OF **THE HARVARD GROUP**

SYNTAX ZX80 is a monthly newsletter exclusively for ZX80 and MicroAce owners. We bring you news, reviews and applications for your computer, plus technical notes for circuit-builders. SYNTAX also provides a forum for thousands of users to share advice and problems about programs and vendors. We bring you timely updates about new hardware, software and books. And we cover *all* the Sinclair-MicroAce computers, including the new ZX81.

At SYNTAX we emphasize practicality. You can apply our suggestions even if you aren't sure at first why they work, because we give you complete instructions. Text is clear and easy to understand. SYNTAX readers already know about:

- An automatic phone-dialer they can put together in a few hours
- Syntactic Sums™ to check input for errors
- Printing characters four times normal size
- Programs to explore computer memory
- Cassette eavesdropping to locate files on tape and simplify loading
- How to build their own external additional RAM
- How to add an 8212 I/O chip to control external devices from their computers

And SYNTAX readers like what they get every month. Subscribers know they can depend on us.

After receiving only three issues of SYNTAX ZX80, I find that I anxiously await the next issue . . . keep up the good work!

Martin Irons
Goshen, NY

Congratulations on the brass-tacks, down-to-earth approach of your newsletter. I'll be looking forward to future issues.

Otis Imboden
Washington, DC

Many readers get their first issue and immediately order the back issues — more proof that they like what they see.

What's special about our publication? Just look through one issue. We work hard to bring you a quality newsletter. We strive to print useful programs of above-average accuracy. As any computer magazine editor can tell you, program listing accuracy is tough to achieve, but we boost our average with every issue. We test each program to make sure it works, it fits in the designated RAM, and it runs when you follow the directions. We print program listings in screen-image format to make it easier for you (it's sure not easier for us!) to enter programs accurately. We invented Syntactic Sum™ as an additional aid for you in getting error-free programs. With your subscription you also get access to thousands of other readers, and our staff experts are available by phone to answer your questions or help you solve problems with your machine.

SYNTAX readers get every month:

- Latest news of Z80 hardware and software
- Programs to organize information, calculate, entertain, or instruct
- Do-it-yourself additions to the ZX80/MicroAce
- Clear explanations for beginners

To share the benefits of SYNTAX ZX80, just complete the coupon below and return it with your choice of payment. You will receive a year's subscription, 12 issues, for only $25 in US funds (plus $14 for foreign airmail if you live outside North America).

We are so sure you'll find SYNTAX useful that we promise to refund your entire subscription fee if you aren't satisfied. An unconditional guarantee — you can't lose. But if you're still skeptical, send $1 for a sample issue and see for yourself how SYNTAX can help you use and enjoy your ZX80 or MicroAce more.

Join the others who stretch the ZX80s and MicroAces to their utmost. Act now — as soon as we receive your coupon with payment, your first issue will be on its way. For faster service, phone your credit card order to 617/456-3661. Don't miss SYNTAX!

THE HARVARD GROUP
Bolton Road, Harvard, Mass. 01451

YES! Please send me 12 issues of SYNTAX for $25.

☐ My check for $25 is enclosed.
Make checks payable to:
"The Harvard Group."

☐ Please charge my ☐ MasterCard
☐ VISA ☐ American Express
☐ Diner's Club account.

account number _____

exp. date _____ bank number (MC only) _____

signature _____

I own a ☐ Sinclair ZX80
☐ MicroAce computer.

Name _____ Title _____

Organization _____

Address _____

City _____ State _____ Zip _____

Day Phone ( ) _____ Evening Phone ( ) _____

S1181