November/December 1981 Volume 1, Number 6 $2.50 (USA)
£1.20 (UK)

The magazine for Sinclair users

SYNC®

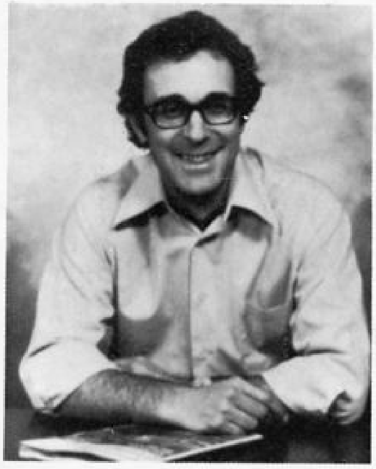
Programs & Hardware:
- Inventory
- Memory & I/O Expansion

Machine Language:
- Introduction
- READ on the ZX80

Features:
- CompuKid
- Perceptions
- Kitchen SYNC
- Resources

Games:
- Word Search
- Taxman
- In a Maze
- Moving Artillery
- Hampson's Plane
- Hidden Chessmen

A REMARKABLE MAGAZINE



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 light-hearted 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.

Term	*USA*	*Canada and 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

Mr Ebbs

The magazine for Sinclair users

SYNC

November/December 1981

Volume 1, Number 6

DEPARTMENTS

MACHINE LANGUAGE

APPLICATIONS

GAMES

Staff

Publisher/Editor-in-Chief	**David H. Ahl**
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**
Personnel and Finance	**Patricia Kennelly**
Customer Service	**Ralph Lovery**
Order Processing	**Ruth Coles**
Circulation	**Frances Miskovich** **Carol Vita**

MEMBER MITA

Index to Advertisers

Blank Cassettes	37
Books for the Sinclair	15
Burnett Electronics	13
Byte-Back	5
Campbell Systems	41
Computer Cartoon Book	29
Computer Coin Games	43
Creative Computing subscriptions	cover 2
D. Bruce Electronics	39
Richard Francis	40
Gladstone Electronics	11
Hardware Haven	41
Insight	39
J. Edmonds	28
JRS Software	7
Katie and the Computer	29
Lamo Lem	9
L.J.H. Enterprises	31
Sinclair Research	24, 25
Small Business Computers subscriptions	cover 4
Softsync	3
SYNC subscriptions	17
T-shirts	cover 3
Zeta Software	31

Volume 1, Number 6

SYNC (USPS: 585-490: ISSN: 0279-5701) is published bi-monthly for $16 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.

Subscription rates: USA: 6 issues $16; 12 issues $30; 18 issues $42. Canada and foreign surface: 6 issues $20; 12 issues $39; 18 issues $56. UK air: 6 issues £13; 12 issues £25; 18 issues £36. Other air: 6 issues $31; 12 issues $60; 18 issues $87. Call (800) 631-8112 toll-free (in N.J. 201-540-0445) to begin your subscription.

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

The Cover
The cover shows scenes from the first ZX Microfair which was held in London in September. Photos courtesy of Richard Gollner of Radala & Associates in London.

letters

Thick Black Bars

Dear Editor:

Thick black bars on the display screen may be caused by 60 Hz. A.C. hum resulting from a failing capacitor in the power supply. Cure is replacement. On my MicroAce this involved breaking open the external power supply case at the glue lines, and replacing the large 1000 M.F.D. capacitor with a new one. Be sure to observe the polarity if you make this replacement.

Cecil Bridges
1248 N. Denver
Tulsa, OK 74106

Help Wanted

Dear Editor:

Do you think that a receive only teletype program could be written into the 1K of RAM or would it take more?. . .

Thanks for producing a fine magazine for a pretty neat little computer.

James S. Johnson

Dear Editor:

I have had my ZX80 micro computer for a few months now and I must say I am very pleased with it. The only drawback to the system is its limited RAM. I know that a 16K RAM pack is available, but I feel I do not need that much. I would be satisfied with 2K or even 4K.

I am very familiar with electronics and have done extensive breadboarding of digital projects so I would like to see an article showing a do-it-yourself memory expansion with schematic diagram and all the details.

Eric Bergstrom
3957 Denley, Apt. 108
Schiller Park, IL 60176

Dear Editor:

I have a Sinclair ZX80 with 8K ROM and 16K RAM. Can you suggest where I can get information on how to:

1) Use the ZX80 as a terminal with coupler.

2) Decode and display Morse Code and RTTY from shortwave (see QST magazine, July 1981, p. 30).

3) Connect external inputs such as switches or analog inputs to connector or ear-jack.

4) Connect a printer (Is the ZX8l printer available?).

5) Implement "slow" display mode.

Ronald Silver
2635 Cranston Rd.
Philadelphia, PA 19131

Ed.—Readers who have suggestions for meeting the above requests are invited to send their ideas to **SYNC** *letters. The ZX printer will not be available yet in the US market although it is available in the UK.*

Scrolling REM Statements

4K ROM

Dear Editor:

When entering a machine language program into a REM statement, you want to scroll the REM off the screen. After entering the Basic program, use the immediate instruction POKE 16403,A. A should be the next Basic statement number after the REM statement. Poking this puts statement A at the top of the screen. I have found this much easier than using dummy statements, and I have not found this technique suggested elsewhere.

Richard Van Workum
920 Leslie Ln.
Hanford, CA 93230

Cecil Bridges' LED Load Monitor

Dear Editor:

Cecil Bridges' article entitled "Adding an LED Load Monitor to the ZX80" (*SYNC* 1:1) actually describes this useful hardware modification for a MicroAce and not a ZX80. If a schematic were available for this modification to a ZX80, it would be greatly appreciated since the MicroAce diagram is useless to me.

Also, no part numbers are listed for the Radio Shack LEDs that have been used or at least their low current requirements.

Hopefully you can provide me with the necessary information.

Cal Butler
81 Dorian Lane
Rochester, NY 14626

Ed.—David Ornstein points out that Cecil Bridges' LED Load Monitor can be adapted to work with the ZX80. Although the title implied that the circuit was for use with a ZX80, the connection diagram given was for the MicroAce, as Reader Butler correctly notes. To use the circuit with the ZX80, connect the X wire as shown in the article. The Y wire should be connected to ground. A good place to tap ground is just below IC17, on the large silver pad on the printed circuit board.

sync notes

Paul Grosjean

ZX Microfair Report

September 26, 1981, was a great day for many ZX80 and ZX81 users in the UK. They braved the wind and rain to go to the first ZX Microfair, held in Central Hall, Westminster, London. The show was a resounding success to the extent that some people queued up to one hour to get in. But once inside, the display of new software and new products was well worth the wait.

Quite a few new products caught my attention. First of all, Quicksilva had a Programmable Character Generator Board which allows you to create your own character set, either for output to the television or to the ZX Printer. The board would, therefore, allow an upper and lower case set to be generated or a "Space Invader" character to be formed. Also on Quicksilva's stand was their Sound Board playing Bach in three-part harmony and a full feature *Defender* program with sound effects (if a Sound Board is connected, that is).

Moving on, DCP Microdevelopments has a prototype Voice Synthesizer Board attached to a ZX81 through one of their Peripheral Packs. The Voice Synthesizer should be available in early 1982, but the Peripheral Pack, containing 4K RAM and an 8-Bit Input/Output Port, is available now.

Technomatic had a demonstration of some of the many uses for their low cost I/O Port for the ZX80 and ZX81, including music (well actually just a series of bleeps!) and various external control functions.

Haven Hardware had their Programmable Character Generator on show and a prototype of their Colour Board. The Colour Board looks very impressive, but is not available yet.

Lots of software cassettes were for sale, ranging from Business Database packages to arcade-game programs. There must have been some fifty different software packages and fifteen books about the ZX80 and ZX81!

Sinclair had one of the new ZX Printers at the Microfair, but no other product news at the moment. The range of Sinclair software was there for sale, as was the ZX81 and 16K RAM Pack, but not the ZX Printer.

SYNC was there, represented by Hazel Gordon, Creative Computing's UK representative. Altogether 32 exhibitors crammed into the one hall, and, despite the problem of large crowds, a fantastic time was had by all who went.

Martin Wren-Hilton
UK Correspondent to SYNC

ZX81 Launched in the U.S.

Sinclair Research Ltd., introduced the ZX81 to the U.S. market on October 7, 1981, at a press conference in Boston, Massachusetts. The ZX81 succeeds the ZX80. Based on Sinclair's innovative four chip design, the ZX81 will be the least expensive personal computer in the world. It will retail in the U.S. at $149.95 assembled and at $99.95 for the kit version. Mail order sales will continue to be the main means of distribution, but American Express will also act as distributor.

Over 100,000 have been sold in Britain since its introduction there in March 1981. Sinclair expects the unit sales of the ZX81 to pass the unit sales of all other personal computers by the end of 1981.

In the question period following the announcement, Clive Sinclair emphasized that the ZX81 is aimed at two main potential buyers: the hobbyist and the man on the street who wants to learn about computers. Sinclair Research will emphasize educational programs and provide software. Software from others will be encouraged. Asked about a disc system, Sinclair noted that a British firm is offering one already and that Sinclair Research is working on its own model to be available sometime next year.

The ZX81 is described in Sinclair Research's ad elsewhere in this issue of *SYNC*, so we will not repeat that information here. But three features that will be of great interest to U.S. users are: a switch allowing use on either channel 2 or 3, a built-in booster circuit for tape loading, and an increased power supply that will accommodate the 16K RAM.

Most impressive, however, was the printer demonstrated along with the ZX81. Unfortunately, it will not be available in the American market for the immediate future. A full sized sample of the printout demonstrated is shown in Figure 1.

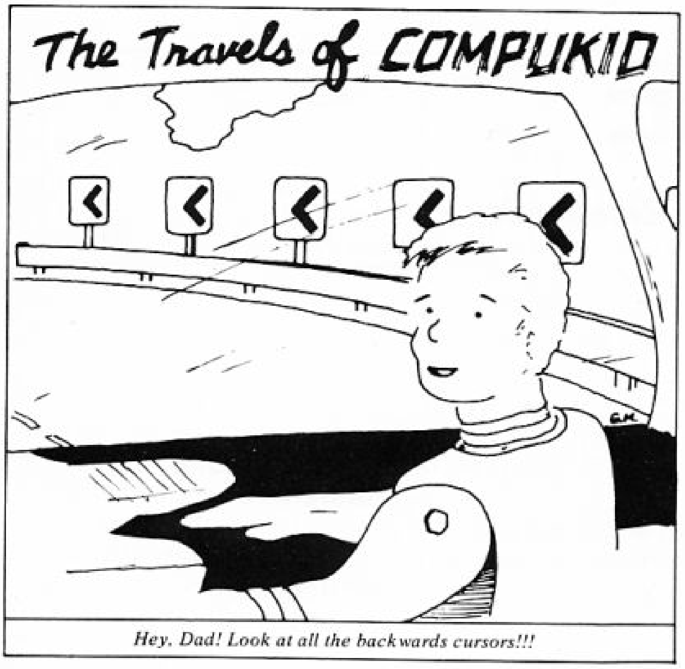
The World of Compukid

We have had a generation grow up not knowing a world without radio, another not knowing a world without TV, another not knowing a world without computers. Now we are watching another growing which has not known a world without personal home computers. When we look at the world, we use the images and the symbols we grew up on. This generation will look at the world with "computer eyes" and see things in new ways. We want to capture something of what they see and share it through the cartoon series we are introducing in this issue of *SYNC* "The World of Compukid [com-pu-kid]." How will the "computer age kids" use the computer symbols and language to describe and interpret the world? The difference between the compukid's picture and the more conventional one should bring a smile at least and hopefully a good laugh.

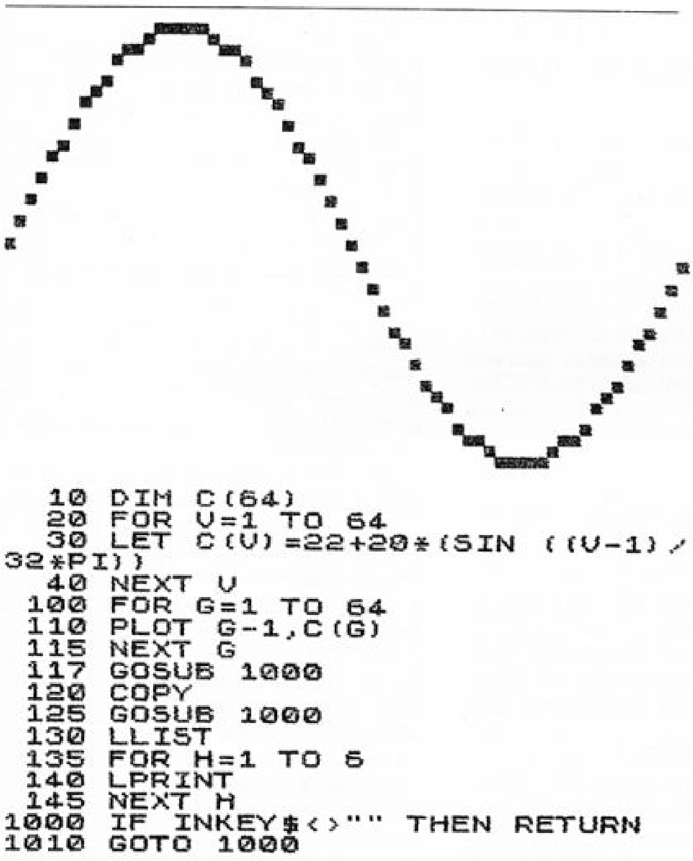
We invite you (especially our younger readers) to submit ideas for our cartoonist. Natural, real life incidents are the best, but you do not need to limit your contribution to those. Older readers may have some fun by trying to see how the world might look through "Compukid's" eyes.

Second ZX Microfair to be Held

Mike Johnston, who organized the first ZX Microfair held in September, has announced that a second one will be organized for January 30, 1982, again at Central Hall Westminster, London. Extended hours and doubled floor space will take the second fair beyond the first. Anticipating a largër show, Mike still hopes "to keep the same informal and friendly atmosphere" of the first. Mike can be contacted at 71 Park Lane, Tottenham, London, N17 OHG (Tel. after 7 p.m., 01 801 9172).



Hey, Dad! Look at all the backwards cursors!!!



```
  10 DIM C(64)
  20 FOR V=1 TO 64
  30 LET C(V)=22+20*(SIN ((V-1)/
32*PI))
  40 NEXT V
 100 FOR G=1 TO 64
 110 PLOT G-1,C(G)
 115 NEXT G
 117 GOSUB 1000
 120 COPY
 125 GOSUB 1000
 130 LLIST
 135 FOR H=1 TO 5
 140 LPRINT
 145 NEXT H
1000 IF INKEY$<>"" THEN RETURN
1010 GOTO 1000
```

Figure 1.

4K ROM
1K RAM

Kitchen Sync

Alan Groupe, Michael Tardiff, and Ivan Zatkovich

Making the Most of What You've Got

This month's column was inspired by two other articles appearing in the May/June issue of *SYNC*. In the first, David Lubar told of his experiences in using the NOT operator of Sinclair Basic. In the second article, Bill Eckel introduced a game called *Black Hole* and presented a version which he "almost" got to fit in the ZX80's 1K available memory. These two articles started us thinking, and we decided that the *Black Hole* game might be the perfect medium for demonstrating both the usefulness of AND, OR, and NOT, as well as a method for packing as much possible into what might be considered a "tiny" amount of memory.

But first, a short discourse on logical operations.

As you have heard over and over again, computers only understand "ones and zeros." While this is true, a computer's life is not quite that dull. Since you cannot do much with just 0 and 1, bunches of 0's and 1's (called bits) are grouped together into larger sets, the most common of which are bytes.

The normal arithmetic operators you are accustomed to (+,-,*,/) take integers (which in the ZX80 are represented as groups of 16 bits) and add, subtract, multiply, or divide them as numbers. Likewise, AND, OR, and NOT take integers and combine them in various ways. In the instance of these operators (called logical, or Boolean, operators), however, it is more useful to think of the integer as the group of 16 individual bits rather than as a whole.

The operation "A AND B" is the collection of 16 bits that are the result of each bit of A "ANDed" with each bit of B. These logical operators are often shown in the form of tables as follows:

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

NOT	
0	1
1	0

As you can see, AND returns a 1 when both bits are 1; OR returns a 1 when either bit is equal to 1. NOT inverts the bit value (returns 1 if given 0, 0 if given 1).

There is one more Boolean operator you should know about, called the EXCLUSIVE OR, or XOR for short. Here is what it looks like:

XOR	0	1
0	0	1
1	1	0

While OR returns a 1 if either or both bits are 1, XOR returns a 1 only if either, but not both, are 1. In other words, XOR returns a 1 if the bits are different. While XOR is not in Sinclair Basic, it can easily be coded as:

(A OR B) AND NOT (A AND B)

Groups of bits are combined to create larger values much the same way as decimal digits are combined in base 10. In base 10, the digit positions represent increasing powers of 10 from right to left (1, 10, 100, and so on). The value of the number is the sum of the digit values. Therefore, "24" represents 2 tens and 4 ones, added together, which is 24.

Binary works the same way. Bit positions represent increasing powers of 2 from right to left (1, 2, 4, 8, 16, etc.). Hence, "11000" represents 1 sixteen, 1 eight, and no fours, twos, or ones. Adding these together, we also end up with 24.

Specific bits can be turned on or off using the AND and OR operators of Sinclair Basic. To turn a bit on in an integer, you can "OR" it with a word that has just that bit (or bits) turned on. To turn a bit off, "AND" the word with one in which every bit except the one you want is turned on.

To see how all this works, try the test program in Listing 1. It asks for two numbers, A and B. Enter these in decimal (as normal numbers). You should keep running the program until you have some understanding of what is going on.

The main program itself is rather simple. The subroutine at line 1000 probably deserves some discussion, though.

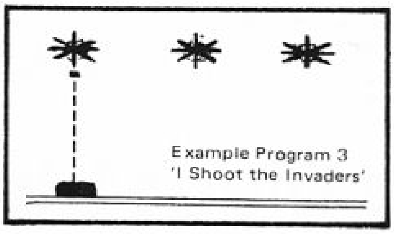
Lines 1000-1010 test the leftmost bit (the sign bit, which indicates if the integer is a positive or a negative number) and print out a 0 or 1 depending on that bit's value. The remaining bits can be tested by bit position, using a loop that steps from 14 down to zero. (The leftmost bit could not be tested in this loop because of the way the Sinclair handles numeric overflow.) Lines 1020-1030 and 1060-1070 set up a FOR-NEXT loop that counts down from 14 to 0. Lines 1040-1050 test the single bit 2**Y1 (Y1 going from 14 to 0 – 2**0 = 1)

ZX80

JRS SOFTWARE

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

ZX81



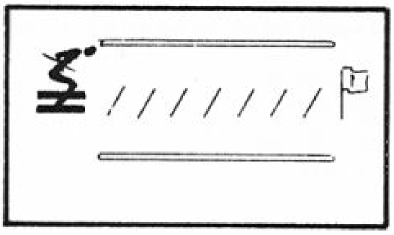
ZX80 - PROGRAMMABLE MOVING DISPLAY

(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.

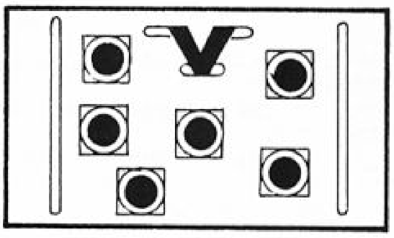
Cassette with 1k, 2k versions and 3 example programs plus FULL documentation **£4.95**



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.

As you can see, AND can also be used to test specific bits. If you "AND" the integer you want to test with another in which only the bits you want to test are equal to 1 (called a "mask" by programming types) the integer you end up with will be zero if none of the tested bits were set, and non-zero if any were. The resulting integer will have a 1 in each bit position that was equal to 1 in both the tested integer and the mask.

Now, back to *Black Hole*. If you remember, the problem was that the program still did not fit into a 1K Sinclair, which kept most Sinclair owners from playing the game. Not being very happy about being deprived of some fun with our ZX80s, we tried to do something about that.

Let us take a good look at the game itself. The game board, called the "galaxy," has nine positions, each of which can be either a "star" or a "black hole." The most obvious way to store the galaxy is with an array of nine integers, each representing a position in the galaxy. Since there are only 9 positions and since each position can only be in one of two states (star or hole), the entire galaxy can be represented in a single integer, with 9 of the bit positions representing the nine positions in the galaxy. This type of representation saves memory in two ways—the game board itself takes less memory, and, more importantly, the entire galaxy can be updated all at once, rather than by twiddling each position separately.

Listing 1.

```
  1Ø PRINT "ENTER A"
  2Ø INPUT A
  3Ø PRINT "ENTER B"
  4Ø INPUT B
  5Ø CLS
  6Ø PRINT "A",
  7Ø LET Z=A
  8Ø GO SUB 1ØØØ
  9Ø PRINT "B",
 1ØØ LET Z=B
 11Ø GO SUB 1ØØØ
 12Ø PRINT "##NOT A",
 13Ø LET Z= NOT A
 14Ø GO SUB 1ØØØ
 15Ø PRINT "#NOT B",
 16Ø LET Z= NOT B
 17Ø GO SUB 1ØØØ
 18Ø PRINT "A AND B",
 19Ø LET Z=A AND B
 2ØØ GO SUB 1ØØØ
 21Ø PRINT "A OR B",
 22Ø LET Z=A OR B
 23Ø GO SUB 1ØØØ
 24Ø PRINT "A XOR B",
 25Ø LET Z=(A OR B) AND NOT
     (A AND B)
 26Ø GO SUB 1ØØØ
 27Ø PRINT
 28Ø PRINT
 29Ø GO TO 1Ø
1ØØØ IF Z<Ø THEN PRINT "1";
1Ø1Ø IF NOT Z<Ø THEN
     PRINT "Ø";
1Ø2Ø LET Y1=14
1Ø3Ø FOR Y=1 TO 15
1Ø4Ø IF Z AND 2**Y1 THEN
     PRINT "1";
1Ø5Ø IF (Z AND 2**Y1)=Ø
     THEN PRINT "Ø";
1Ø6Ø LET Y1=Y1-1
1Ø7Ø NEXT Y
1Ø8Ø PRINT "#";Z
1Ø9Ø RETURN
```

The representation we chose uses bits 1-9 (bit numbers go from 15 on the left to 0 on the right) to represent positions 1-9. So the bit that represents the state of position 'n' is 2**n. The remaining bits are zeros. Since the initial state of the board is all holes (0) with a star (1) in position 5, we can set the whole galaxy to the initial state with the statement LET GALAXY=2**5 (line 10). Compare this with lines 10-20 in the original program.

Lines 35-70 simply print out the galaxy.

Line 75 tests to see if the value of GALAXY is 0. GALAXY being equal to 0 means that the collection of 16 bits that represents the value of GALAXY is all zeros. So GALAXY=0 means that each position in the galaxy is a black hole—the definition of losing.

Line 95 tests to see if the value of GALAXY is 990. This value is represented by a pattern of bits where bits 1, 2, 3, 4, 6, 7, 8, and 9 are 1's and bit 5 along with the unused bits (0 and 10-15) are zeros. You can verify this with the binary calculator above. This pattern is the definition of a win.

Line 130 tests to see if the selected position is valid. Since you can only shoot at stars and not at black holes, the input would be invalid, if the selected bit position were a zero.

Lines 135 and 202-219 are the most cryptic part of the program. Looking back at Eckel's original *Black Hole* article, we see that for each possible move, there is a pattern of positions that must be inverted (a star becomes a hole, a hole a star). If the state of the galaxy can be represented in a single integer, then certainly each of these patterns can be too. Line 135 selects the appropriate subroutine to set the value of CHANGE to represent the required pattern.

For example, if you look at the pattern of positions that must be inverted if you choose to shoot at star 5, you can see that positions 2, 4, 5, 6, and 8 must be inverted. The others must stay the same. This can be represented by setting the corresponding bit for each position that must be inverted. Therefore, the required value (mask) would have bits 2, 4, 5, 6, and 8 set with the other bits cleared. This is what is done in line 210. You can use the binary calculator program to verify that 372 has the correct pattern of zeros and ones.

While OR can be used to set certain bits to ones and AND can be used to set certain bits to zeros, XOR can be used to invert certain bits without knowing their original value. This is done in line 140.

As you can see, by carefully selecting the representation of the data of your program, you can squeeze more into—and get more out of—the 1K memory of the Sinclair.

```
 1Ø LET GALAXY=2**5
 15 CLS
 2Ø PRINT "BLACK HOLE"
 25 PRINT
 3Ø PRINT
 35 FOR I=Ø TO 2
 4Ø FOR J=1 TO 3
 45 IF (GALAXY AND (2**(3*
    I+J)))=Ø THEN PRINT "O ";
 5Ø IF GALAXY AND (2**(3*
    I+J)) THEN PRINT "* ";
 55 NEXT J
 6Ø PRINT ,3*I+1;" ";3*I+2;
    " "; 3*I+3
 65 PRINT
 7Ø NEXT I
 75 IF NOT GALAXY=Ø THEN GO
    TO 95
 8Ø PRINT "YOU BLEW IT"
 85 PRINT "YOU ARE LOST IN
    SPACE FOREVER"
 9Ø STOP
 95 IF NOT GALAXY=99Ø THEN
    GO TO 115
1ØØ PRINT "CONGRATULATIONS"
1Ø5 PRINT "YOU FOUND THE
    BLACK HOLE"
11Ø STOP
115 PRINT "WHICH STAR?"
12Ø INPUT STAR
125 IF STAR<1 OR STAR>9 THEN
    GO TO 12Ø
13Ø IF (GALAXY AND 2**STAR)
    =Ø THEN GO TO 12Ø
135 GO SUB 2*STAR+2ØØ
14Ø LET GALAXY=(GALAXY OR
    CHANGE) AND NOT (GALAXY
    AND CHANGE)
145 GO TO 15
2Ø2 LET CHANGE=54
2Ø3 RETURN
2Ø4 LET CHANGE=14
2Ø5 RETURN
2Ø6 LET CHANGE=1Ø8
2Ø7 RETURN
2Ø8 LET CHANGE=146
2Ø9 RETURN
21Ø LET CHANGE=372
211 RETURN
212 LET CHANGE=584
213 RETURN
214 LET CHANGE=432
215 RETURN
216 LET CHANGE=896
217 RETURN
218 LET CHANGE=864
219 RETURN
```

Glitchoidz Report

Widget (1:2, p. 23)

Author McGath writes: "Wait one widgeting minute! It is not necessary to tinker with the economics of *Widget* to make it winnable; just correct one typo: 600 should be
LET S=S+B*8
This vastly improves the return on small advertising outlays as was originally intended. The other tinkering can still be done to vary the challenge of the game, but a conservative strategy will result in steady growth. I hope the error in *Widget* has not shaken anyone's confidence in the free market too badly."

Looking inside the ZX80 (1:3, p. 16)

```
92 PRINT CHR$(X+28);" ";
240 PRINT CHR$(X);
```

A Trick and a Graphic System (1:3, pp. 30-31)

Author Comer calls attention to the continual problem of making a distinction between zero and the letter O. Since this is not clear in the listing, he suggests: "In order to get the examples to run all one need realize is that everything is an Oh. The only exception is in the example called the 'U.S. Map.' At the end of the second line the data should be "eleven Oh twelve." Otherwise the map data is correct, but pay close attention to the zeros (which are narrow) and the Ohs (which are wider)."

Perceptions (1:5, p. 11, col. 2)

4th line from bottom: R2 should be R32.

Gra+PIX (1:4, p. 16, listing 5)

Add:

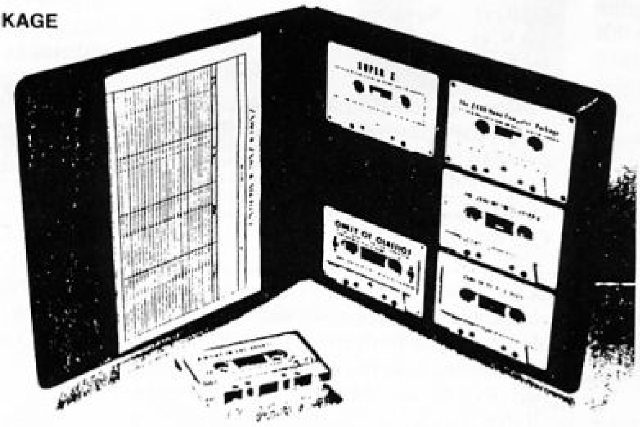
```
9010 REM ENTER FROM POLYGON/SEGM
ENT/ARC
```

Correct:

```
9920 IF P2>2*PI THEN LET P2=P2-2
*PI*INT (P2/(2*PI))
9925 IF T2>2*PI THEN LET T2=T2-2
```

Hangman (1:4, p. 40)

340 Be sure to use zero after *G*
430 PRINT CHR$(CODE(C$));"#";



4K ROM
8K ROM

perceptions

David Ornstein

Conversions: 4K ROM to 8K ROM and 8K ROM to 4K ROM

One of the big questions raised by Sinclair users is how to convert programs from the 4K ROM to the 8K ROM and vice versa. This article will serve as a comprehensive guide to such attempts. Let's begin by a review of each of the commands and functions of both ROMs. It should be noted in the interim that there is no guarantee that all programs can be converted from one ROM to the other. It should also be noted that, barring an extremely complex software scheme, there is *no way* to load a 4K program from tape into an 8K machine or 8K programs on tape into a 4K machine.

Expressions

An expression is a series of values, strung together with operands, for example, 3-4+8/AMD. Expression evaluation is carried out by applying operators to their operand(s). For example, applying the + operator to the operands 4 and 26 yields the value of 30. Expressions are evaluated, operator by operator, according to the priorities of the operators they contain. The priorities for operators are shown in Figure 1.

As you can see, the only difference between the ROMs is the priority of the / (division) operator. In theory, this could cause problems. In practice, however, one can usually be assured that an expression specified on one machine, can be transferred as is, without ill effects. If you do find this difference to be a problem, you can always use parentheses to alter the order of evaluation.

Functions

Many functions previously unavailable to the 4K ROM user are now available for owners with the 8K floating-point ROM. The primary problem with converting programs from 8K to 4K is that there is simply no way to work with floating-point numbers (that is, a number like 4.827736, as opposed to an integer like 6 or 327), on the 4K machine. Many of the new functions are "floating-point functions." For example, there is no purpose in having a PI function on an integer-Basic machine, as the value 3 is not as useful as 3.14159265.... The following functions fall under this category of "floating-point only":

ACS(x)	Arc Cosine
ASN(x)	Arc Sine
ATN(x)	Arc Tangent
COS(x)	Cosine
EXP(x)	Natural logarithm (base e)
PI	The value 3.14159265
LN (x)	Natural logarithm
SIN(x)	Sine
SQR(x)	Square root
TAN(x)	Tangent
INT(x)	Integer

Many other functions, however, are worth the conversion process. These are detailed below.

CHR$(x)
CODE(x$)

These functions perform the operation of converting a number to its corresponding character, and vice versa. They work identically on both ROMs, but it must be emphasized that the character sets on the two ROMs are different. For example, CHR$(20) yields a multiplication sign on a 4K ROM, but on an 8K ROM it produces an equals sign. You must not assume that a line such as:

```
3100 IF CODE(A$)=13 THEN GOTO
100
```

can be directly transferred to the other ROM. Most of the character set, primarily the most important and most commonly used sections, are the same on both ROMs. The characters for numbers and letters have the same codes, but this is not true of all characters. The primary rule is: *If you are not sure, check it.*

RND
RND(x)

The random number generation functions differ simply in that the 4K returns a random integer and the 8K a random floating point number. The 4K random number function, when called, spews up a random integer between 1 and x, x being the argument of the call, i.e., J=RND(x). The 8K function RND(no argument) returns an FP (floating-point) value between 0 and 1. It is clearly a fruitless venture to attempt to simulate the 8K function on a 4K machine, as you cannot have a floating point number in an integer Basic program. (A number between 0 and 1 must, obviously, be an FP number.)

Operator	8K ROM priority	4K ROM priority
Substrings	12	—
All functions (except unary minus and NOT)	11	11
**	10	10
Unary Minus	9	9
*	8	8
/	8	7
+	6	6
- (binary)	6	6
=,<,>	5	5
<=,>=,<>	5	—
NOT	4	4
AND	3	3
OR	2	2

Figure 1.

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. . . .
$9.95 ($12.95 in Canada)

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)

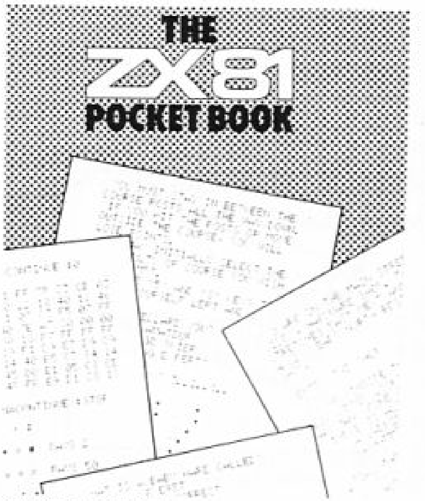
Exciting Book Titles!

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

It is, however, possible to obtain a random integer on an FP machine. To get a random number between 0 and x-1, use the expression INT(RND*x)+1. Compare the sample lines below:

4K Integer Basic

```
100 J+RND(100)
110 REM set J equal to a random
120 REM number between 1 and 100
```

8K FP Basic

```
100 J=INT(RND*100)+1
110 REM set J equal to a random
120 REM number between 1 and 100
```

INKEY$

The INKEY$ function is available only on the 8K ROM. Without a moderate amount of machine language programming, there is no way to simulate it on the 4K machine. If the thought of such an endeavor strikes you as an interesting project, I would suggest my article on the ZX80's keyboard system (*SYNC* 1:2) as a good starting point.

LEN(x$)

This function is not available on the 4K machine, but it is easy to simulate by a simple subroutine. On a 4K machine you would add this subroutine at the end of your main program. Then to find the length, in bytes (characters) of a string you can set a few argument variables and call the subroutine. This process is illustrated in Figure 2. The Len subroutine is found in Figure 3.

SGN(x)

The SGN(x) function is used to determine the sign (not sine) of a given number, its argument. If the value passed to the SGN(x) (often pronounced 'signum') function, is negative, the function returns a value of -1. If the value passed is 0, the function returns a zero. If the value passed is positive, that is, greater than zero, the signum function returns the value -1. For example, if you PRINT SGN(-237) the computer will dislay a -1 on the screen. Whereas, were you to PRINT SGN(237), the computer would place a 1 on the screen.

This is a useful function. It is not quite so useful, however, if you have to use a subroutine call to access it. To elucidate, you lose much of the value of being able to specify an expression as

```
100 LET X=SGN(RES23/2)+USR(AD)*
4+1
```

when you must express it as:

```
100 LET TEMP=RES23/2
110 GOSUB 9000
120 LET X=TEMP+USR(AD)*4+1
```

```
9000 REM LEN
         THIS SUBROUTINE FINDS
9010 REM THE LENGTH OF A STRING.
         TO USE IT, SET Z$=x$; x$
9020 REM BEING THE STRING YOU
         WANT TO FIND THE
9030 REM LENGTH OF. THE SUB-
         ROUTINE WILL RETURN
9040 REM WITH THE LENGTH OF THE
         STRING IN THE VARIABLE
9050 REM J. YOU SHOULD BE SURE
         THAT YOUR PROGRAM
9060 REM WILL RUN PROPERLY IF
         THESE VARIABLES (Z$,J)
         ARE DESTROYED.
9070 LET J=0
9080 IF Z$="" THEN RETURN
9090 LET J=J+1
9100 LET Z$=TL$(Z$)
9110 GOTO 9080
```

Figure 2.

```
100 REM MAIN PROGRAM
110 LET H$="LKJHGTYL"
120REMNOWCALLTHELENROUTINE
130 LET Z$=H$
140 GOSUB 9000
150 PRINT J
160 REM THIS PROGRAM WILL
PRINT
170 REM THE NUMBER 8, BECAUSE
H$
180 REM HAS 8 CHARACTERS IN IT.
190 STOP
```

Figure 3.

The listing of the routine to simulate SGN(x) is given in Figure 4. The calling procedure for the SGN(x) subroutine is found in Figure 5. You may, however, find it more useful (and easier) to find the signum, directly in your program, as opposed to calling it as a subroutine all the time.

```
9000 REM SGN
9010 REM THIS SUBROUTINE IS USED
9020 REM TO FIND THE SIGNUM OF
9030 REM A GIVEN NUMBER. TO USE
9040 REM IT, YOU PASS ITS ARGU-
9050 REM MENT IN THE VARIABLE
9060 REM TEMP. THE SUBROUTINE
9070 REM RETURNS THE SIGNUM OF
           TEMP IN TEMP.
9080 IF TEMP=0 THEN GOTO 9200
9090 IF TEMP>0 THEN GOTO 9230
9100 REM HERE IF TEMP < 0
9110 TEMP=-1
9120 RETURN
9200 REM HERE IF TEMP=0
9210 RETURN
9230 REM HERE IF TEMP>0
9240 LET TEMP=1
9250 RETURN
```

Figure 4.

```
100 REM MAIN ROUTINE
       |
       |
       |
500 LET TEMP=N                ! You now want the SGN(N)
510 GOSUB 9000                !Call the subroutine
520 LET NEWVAL=TEMP           !Get the SGN into NEWVAL
       |                      !
       |                      ! This sequence of code
       |                      ! is equivalent to:
       |                      !
       |                      ! LET NEWVAL=SGN(TEMP)
       |                      !
       |
       |
9000 REM SGN
       |
       |
       |
```

Figure 5.

VAL(x$)

The VAL(x$) function is used to convert between a string containing the character representation of a number to an actual number. For example, PRINTing the VAL("1234") yields the number 1234, *not the string* "1234". You cannot assign the result of using the VAL() function to a string, nor can you use a number as the argument. That is, X$=VAL(T$) is illegal, as is T=VAL(J). Once again, the 4K ROM falls short by not providing this useful function, but we can use the subroutine in Figure 6 to simulate it. The subroutine takes the argument T$ and returns the VAL of it in the variable TEMP.

```
9000 REM VAL
9010 REM
9020 REM THIS SUBROUTINE IS USED
9030 REM TO FIND THE VAL(T$). IT
9040 REM RETURNS THIS VALUE IN
9050 REM THE VARIABLE TEMP.
9060 LET TEMP=0
9070 IF T$="" THEN RETURN
9080 LET TEMP=TEMP+CODE(T$)-28
9090 LET T$=TL$(T$)
9100 LET TEMP=TEMP*10
9110 GOTO 9070
```

Figure 6.

STR$(x)

This function is the inverse of the VAL() function. It returns a string which corresponds to the characters which would appear on the screen if you were to PRINT the number. For example, LETting T$= STR$(23) would give the 2-character string "23", *not the number* 23. This function is available on both the 4K and 8K ROMs, and works identically on both. Therefore, when translating from one ROM to the other, you can use exactly the same function without any problems.

USR(x)

The USR (pronounced 'user') function is used to call machine language subroutines in the computer. The 4K and 8K versions differ only in where they get the value to return to Basic. The 8K version, when your machine language subroutine RETurns, yields the 16-bit number held in the BC register-pair. The 4K version returns the value in the HL register-pair. (Do not worry if you do not understand this; this is directed to machine language programmers.)

ABS(x)

This function returns the absolute value of its argument.This function is available on each ROM and can be transferred freely between 8K and 4K ROM programs.

PEEK(x)

The PEEK function is used to find the value of an arbitrary byte of memory (i.e., byte x). It is available on both ROMs. The only difference between the two is that, whereas on the 4K ROM one might say:

J=PEEK(-13420)

on the 8K ROM one would say:

J=PEEK(46188).

The formulae for conversion between ROMs are:

4K to 8K:

If the address is negative, make it positive and add 32768.

8K to 4K:

If the address is greater than 32768, subtract 32768 and make it negative.

NOT X

This function is used to obtain the logical inverse of the value given as the argument. It is available on both ROMs. The only difference is that on the 4K ROM "True" is represented as -1. Whereas on the 8K ROM, "true" is represented by 1. Both ROMs use 0 to represent "false."

X AND Y/X OR Y

AND is available on both ROMs. It does, however, work differently. On both ROMs a statement such as:

IF A=12 OR B>37 AND X$="Y" THEN GOTO 1000

will execute identially on both ROMs. But a statement such as:

JY=P AND 2+Z

will create different results on the two ROMs.

On the 4K ROM AND (and OR) are binary. That is, each bit of the operands are ANDed (or ORed) together. On the 8K ROM the AND and OR operators work differently. The expression AN AND XZ will yield 0 if XZ is equal to 0. Otherwise, it will return AN. Thus,

47 and 227

returns 227. But,

0 and 227

returns zero. OR works in a similar way. X OR Y returns 1 if Y=0. Otherwise, it returns X. The general rule for conversion is: If the operators are used in a boolean expression (like an IF statement), they can be transferred directly. If not, you are probably in trouble.(*Ed.—See* **Kitchen SYNC** *in this issue for further discussion of AND, OR, and NOT.*)

In the next issue, I will continue the discussion of program conversion by giving tips for converting commands (e.g., PRINT and INPUT) from one ROM to the other. Until next time, same relativistic time period, same non-Euclidian universe. ■

4K ROM; 1K RAM
8K ROM; 1K RAM

How is it done?

An Introduction to Machine Code

Dr. Ian Logan

The managing editor and I are frequently asked about how one starts to use machine code on a ZX80/1. So this article is an attempt to reply to these questions, and I trust that you will find that machine code is not only for the expert.

An Outline View

The ZX80/1 microcomputer system as supplied by Sinclair Research is capable of being programmed in two different languages, i.e., Basic and Machine Code.

Basic is a very easy language to use for the beginner and, as long as one's programs are simple, the language is almost 'ideal.' However, Basic is a rather 'slow' language and limited in its commands.

Machine code, however, is a much more difficult language to use. The resultant programs are executed by the Z80 microprocessor at a fantastic speed and the complexity of the programs is limited more by the knowledge of the programmer than by the actual microcomputer.

It is always difficult to explain to the 'beginner' just how to write a machine code program, but in this article we will begin by drawing upon the similarities between Basic and machine code.

Program Structure

A Basic program is made up of a set of Basic lines. In the ZX80/1 system these lines are kept in an area of the RAM (random access memory) that is termed by Sinclair as the PROGRAM AREA. When the user first turns on the machine, this PROGRAM AREA is empty, and the user will then proceed to enter a program into this area. The program can be as short as a single line, e.g., 10 PRINT or can be several hundred Basic lines. The user will then RUN the program, and this will result in the system interpreting line after line of the program, as has been determined by the programmer, until the 'last' line has been reached.

Ian Logan, 24 Nurses Lane, Skellingthorpe, Lincoln LN6 0TT.

A machine code program is in many ways dealt with in a similar manner. First, the programmer must decide just what part of the RAM he is going to designate as his 'machine code area.' It is possible in the ZX80/1 systems to choose an area from several different parts of the RAM but my favorite technique is to reserve part of the PROGRAM AREA by using a REM statement. The next task is to actually enter the machine code into the RAM and this has to be done by using POKE commands. An actual machine code program entered in this fashion can be made up of just a single instruction or many thousands of instructions. This program is then 'run' by using a USR command which is either a single line Basic program, e.g.,

```
10 LET A=USR(16427)
```

or a USR command occurring in a longer Basic program, in which case the machine code program becomes a 'machine code subroutine' of the Basic program. Note how the USR command has to be followed by a number. This number is the address of the location within the machine code area where the machine code program begins.

Instruction Format

All Basic lines can be described as containing an obligatory 'operator'—the command—and an optional 'operand.' The line

```
10 PRINT
```

contains only the 'operator' PRINT whereas the line

```
20 PRINT A
```

contains the 'operator' PRINT and the part that is to be printed, the 'operand' A. Note how the Basic line has the 'operator' coming before the 'operand.'

This division of a line into an 'operator' and an 'operand' is an essential part of Basic syntax and the ZX80/1 systems with their 'syntax checking' facility ensure that the user has no difficulty remembering to place his 'operators' before his 'operands.'

Just as it is in Basic so it is in machine code, but there are hundreds of different 'operators,' as opposed to the 20 or so in Basic.

Whereas a Basic program is made up of 'decimal numbers and letters,' a machine code program consists of only a set of numbers. These numbers can be considered to be in binary, decimal or hexadecimal arithmetic, but for users of the ZX80/1 systems the use of the decimal values is the easiest method, although the 'expert' will usually only think in hexadecimal arithmetic.

So what are the 'operators' in machine code? Well, they are the decimal numbers 0-255, (hex. OO-FF), but since more than 256 'operators' are required, the numbers 203, 221, 237, and 253 (hex. CB, DD, ED, and FD) introduce a second decimal number into the 'operators.'

In Basic the 'operators' are commonly called the 'commands' and in machine code the 'operators' are called the 'instructions.' Fortunately, one does not have to memorize all the different numbers as each instruction has been given a descriptive 'mnemonic' and most programmers only 'look-up' the numbers when they need them.

The 'operands' in machine code are also numbers in the range decimal 0-255, (hex. OO-FF), and these 'operands' are placed after the instructions proper when they are needed.

A machine code program may also

Make the Most of Your ZX81 or 80

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.

Getting Acquainted With Your ZX81

This book is aimed at helping the newcomer make most effective use of his ZX81. As you work your way through it, your program library will grow (more than 70 programs) along with your understanding of Basic.

The book is chock full of games such as *Checkers* which draws the entire board on the screen. Other games include Alien Imploders, Blastermind, Moon Lander, Breakout, Digital Clock, Roller-Ball, Derby Day, and Star Burst.

But the book is not all games. It describes the use of PLOT and UNPLOT, SCROLL, arrays, TAB, PRINT AT, INKEY$, random numbers and PEEK and POKE. You'll find programs to print cascading sine waves, tables and graphs; to solve quadratic equations; to sort data; to compute interest and much more.

Softbound, 5 1/2 x 8", 120 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

contain 'data.' Once again this will be in the form of locations holding decimal numbers in the range 0-255.

All this is better illustrated by the example in Figures 1 and 2.

Note that the machine code subroutine would occupy 5 locations and would be entered by:

```
50 POKE 16427,62
52 POKE 16428,1
54 POKE 16429,198
56 POKE 16430,6
58 POKE 16431,201
```

and 'run' by using:

```
60 LET A=USR(16427)
```

Note: Reserve 16427-16431 first (16514-16518 for 8K).

A Simple Basic Subroutine	Comment
10 LET Z=1	'operator' is 'LET Z', 'operand' is '1'
20 LET Z=Z+6	'operator' is 'LET Z', 'operand' is 'Z+6'
30 RETURN	'operator' is 'RETURN'.

Figure 1.

The Machine Code Subroutine			Comment
mnemonic	Decimal	Hex.	
LD A,+1	62 1	3E 01	Load the 'A' register with the constant 1.
ADD A,+6	198 6	C6 06	Add the constant 6 to the value in the 'A' register.
RET	201	C9	Return to the calling routine.

Figure 2.

Variables v. Registers

In a Basic program there are two different ways of handling variables. The first is to use 'named variables,' e.g., A,B,COUNTER, and this is very much the standard method. However, there is an alternate method that involves the use of ordinary memory locations to which the user will assign values as necessary. This second technique is commonly used in games that use the display file. E.g., if location 16800 is a 'certain point' on the screen, then 'POKE 16800,...' will assign the required value and 'PEEK 16800' will collect the value of the variable.

A machine code program normally uses this second method. That is, the programmer first selects certain locations that he wishes to be filled with 'named variables'; however, these 'names' are only known to the programmer and not the 'computer.'

It is possible though to take the general concept of the 'Basic named variables' a little further and draw a useful analogy between the use of certain Basic variables and the internal registers of the Z80 microprocessor.

In a Z80 microprocessor as used in the ZX80/1 there are many 'registers.' These registers can be considered as 'named variables' in an internal 'variable area.' Each is equivalent to an ordinary memory location in that it can hold a number which has the decimal range 0-255 (hex.OO-FF). The simple registers are the A, H, L, B, C, D, and E registers. The full set of registers is shown in Figure 3.

Although the registers are equivalent to 'one memory location,' there are many times when it is desirable to use a pair of registers that would thereby have the equivalent of 'two locations in memory.' The simplest register pairings are those of the H and L registers, the B and C registers and the D and E registers. Usually these are written as HL,DE, and BC. Such register pairs can be considered to be able to hold numbers in the range decimal 0-65535 (hex.OOOO-FFFF).

The registers of the Z80 can therefore be considered as follows:

The A register is a variable named 'A'.

The H register is a variable named 'H'

and so on for all the simple registers named above. The register pairs can be considered as:

The HL register pair is a variable named 'HL'.

The BC register pair is a variable named 'BC'.

The DE register pair is a variable named 'DE'.

Actual Machine Code Instructions

Now that the analogy has been made, it is possible to use the variables A, H, L, B, C, D, E, HL, DE, and BC to explain the more simple of the 600+ instructions of the Z80 machine code language.

1) *Loading Constants.*

The simplest instructions are those that are used to load a register or a register pair with a 'constant.' For example, in the instruction 'LD A,+dd', the actual code would be two bytes. The first is a decimal 62, (hex.3E), and the second, the value of the constant itself. This instruction can

The Registers of the Z80 Microprocessor

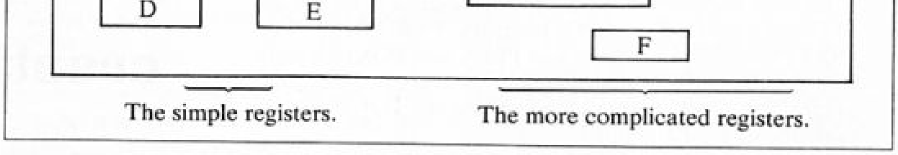


Figure 3.

be considered to have the same result as a Basic line:

LET A+ ...... a constant,

when the variable A is located in the microprocessor. In the instruction 'LD HL,+dddd' the code is three bytes. The first is decimal 33 (hex.21) and the following two are the constant. Note that the constant always appears with the 'remainder' coming before the integer of the 'constant/256.' This instruction would be equivalent to:

LET HL = ... a constant.

or more precisely:

LET L = 'remainder' and
LET H = 'constant/256'

2) Loading Registers from memory Locations

There are only two simple instructions in this group. The first instruction is 'LD A,(addr.)' which is a three byte instruction. The first number is a decimal 58 (hex.3A) and whose other two bytes are the 'address in memory' of the location that is to be copied. Note that the address is once again to be entered as the 'remainder' followed by the 'address/256.'

The Basic equivalent of this instruction is:

LET A=PEEK(1st+2nd,*256)

The other instruction is for loading the HL register pair, and the mnemonic is 'LD HL,(addr.).' Again, this is a three byte instruction. The first byte is decimal 42 (hex.2A), and the other two bytes are the address again.

The Basic equivalent is:

LET HL=PEEK(1st+2nd*256)+256*
PEEK(1st+2nd*256)

or more simply:

LET L= PEEK(addr.)
and LET H=PEEK (addr.+1)

3) *Three Further Instructions*

It is beyond the scope of this article to detail more than just a few of the instructions in the Z80 machine code instruction set, but the following instructions will be used in the game below.

a) The contents of most registers can be copied into another register by using the appropriate instruction.

E.g., the instruction 'LD E,A' copies the contents of the A register into the E register. The instruction code is decimal 95 (hex.5F).

The Basic equivalent would be:

LET E=A

b) The contents of the DE register pair can be added to the contents of the HL register pair by using the instruction 'ADD HL,DE.' This instruction has the code decimal 25 (hex.19).

The Basic equivalent would be:

LET HL=HL+DE

or if preferred:

LET L=L+E
and LET H=H+D+ carry if present.

c) The last instruction of any machine code program must always act as a 'RETURN.' It is easy to understand that this can be performed by the straightforward instruction 'RET' whose code is decimal 201 (hex.C9), but it is often found that the 'return' is made by using a 'stack-handling' instruction instead.

The Basic equivalent of the 'RET' instruction is simply:

RETURN

Once the reader has understood just how instructions are used, it is fairly easy to gradually use the more complex instructions. Suitable lists and tables of all the instructions are to be found in all books on machine code programming, or if the reader prefers he can just take notes on 'new' instructions as he finds them in different programs.

The Demonstration Game

The following Basic program includes many features that can be easily 'machine coded.' However, as with many similar programs, there is no genuine advantage to replacing Basic lines with machine code subroutines, except from the point of interest.

If the reader wishes to try writing the whole of a 'Basic program' in machine code, then I very much suggest that he use a ZX80 with the 8K ROM and a 'slow converter,' or a ZX81, as machine code programming in 'slow' mode is the easiest for larger programs.

The Tower Game

There are two towers of 'bricks' and a single brick is taken from one tower and placed on the other tower. The choice of the 'declining' tower is made at random. The game is over when only a single tower remains.

The larger usable RAM and the compact integer handling of the ZX80 with a 4K ROM allows for a 'more complicated' game than on a ZX80 with the 8K ROM or a ZX81. Therefore, the 8K version is slightly different overall but does use the identical central algorithm.

A First Machine Code Subroutine

The easiest Basic line to convert to machine code is the line:

4K: 230 LET A=PEEK(16396)+PEEK (16397*256

8K: 180 LET A=PEEK 16396+PEEK 16397*256

The Tower Game (8K ROM; 1K RAM)

```
                                          Comment.
 20 RAND                               Different each time.
 30 DIM L(2)                           The two towers.
 40 LET L(1)=62                        The pointers to the tops of the
 50 LET L(2)=69                        towers in the Display File.
 60 FOR A=1 TO 4
 70 PRINT 9-A;"############"           12 spaces.
 80 NEXT A
 90 FOR A=1 TO 4
100 PRINT 5-A;"####A######A"           4sp., graphic A, 6 sp., grap. A.
110 NEXT A
120 PRINT "###77777##77777"            3 sp., 5 grap. "7", 2 sp., 5 grap."7"
130 LET C=0                            Initialise the move counter.
140 PAUSE 50          |FAST            or 140 FOR D=1 TO 20  |SLOW
150 POKE 16437,255    |                -   150 NEXT D        |
160 LET R=INT (RND+1.5)                Randomly choose a tower.
170 LET P=1+(R=1)                      If R=1 then P=2, and vice versa.
180 LET A=PEEK 16396+PEEK 16397        Pick up the start of the D-File.
    *256
190 POKE A+L(R),0                      Remove a brick.
200 LET L(R)=L(R)+14                   Point to "new" top of tower.
210 LET L(P)=L(P)-14                   Point to space above the other tower.
220 POKE A+L(P),8                      Put the brick into place.
230 LET C=C+1                          Count the "move."
240 IF L(R)<118 THEN GOTO 140          If two towers exist, then go back.
250 PRINT "DID YOU GUESS#";C;"#        Did you do well?
    MOVES?"
```

The Tower Game (4K ROM; 1K RAM)

```
                                          Comment.
 20 RANDOMISE                          Different each time.
 30 DIM L(2)                           The two towers.
 40 LET L(1)=62                        The pointers to the tops of the
 50 LET L(2)=69                        towers in the Display File.
 60 FOR A=1 TO 4
 70 PRINT 9-A;"############"           12 spaces.
 80 NEXT A
 90 FOR A=1 TO 4                       The towers.
100 PRINT 5-A;"####A######A"           4 spaces,shifted "A",6 spaces,shifted "A"
110 NEXT A
120 PRINT "###GGGGG##GGGGG"            3 spaces,5 shifted "G".2 spaces,5
130 PRINT                              shifted "G" forms a base to the towers.
140 FOR G=RND(23)+4                    The computers guess.
150 PRINT "YOUR GUESS(4-27)";          Make a guess for the number of
160 INPUT Y                            moves to eliminate a tower.
170 IF Y<4 OR Y>27 THEN GO TO 1        A suitable range. But not the
    60                                 actual range.
180 PRINT "-#";Y;",#MINE-#";G,"        Watch the spacing!
    PRESS N/L"
190 LET C=0                            Initialise the move counter
200 INPUT A$                           Wait for a N/L, before each "move".
210 LET R=RND(2)                       Randomly choose a tower.
220 LET P=1-(R=1)                      If R=1 then P=2, and vice versa.
230 LET A=PEEK(16396)+PEEK(1639        Pick up the start of the D-file.
    7)*256
240 POKE A+L(R),0                      Remove a brick.
250 LET L(R)=L(R)+14                   Point to "new" top of tower.
260 LET L(P)=L(P)-14                   Point to space above the other tower.
270 POKE A+L(P),9                      Put the brick into place.
280 LET C=C+1                          Count the "move."
290 IF L(R)<118 THEN GO TO 200         If two towers exist, then go back.
300 PRINT "IT WAS#";C;"#MOVES A        An end message.
    ND#";
310 IF ABS(G-C)<ABS(Y-C) THEN G        Whose guess was the better?
    O TO 340
320 PRINT "YOU WON"                    Well done.
330 STOP
340 PRINT "I WON"                      Oh dear. Out classed by the ZX80.
```

which picks up the address of the start of the Display File.

In the 4K program one proceeds as follows:

1) Replace line 230 by:
   230 LET A=USR(16427)
2) Enter a line 10
   10 REM 1234

that reserves 4 locations for a machine code routine. The address of the first of these locations being 16427.

3) Remove line 10 from the listing by using, for example:
   LIST 250

and from now on do not use HOME or LIST as this will cause problems. (The problems can be solved by using after a HOME, etc., Cursor down, cursor down & POKE 16403,20 if needed)

4) Enter:

POKE 16427,42	(hex. 2A)
POKE 16428,12	(hex. 0C)
POKE 16429,64	(hex. 40)
POKE 16430,201	(hex. C9)

which will enter a 4-byte machine code routine into the 4 reserved bytes of line 10.

The 'mnemonics' for this program are:

LD HL,(D-File)
RET

where the address of D-File is 16396 in decimal, which splits into a remainder of 12, and 64*256.

5) Run the program. The reason for this routine working is that the expression:

```
USR(16427)
```

returns to the Basic program the current value of the HL register pair.

In the 8K program one proceeds in a similar manner.

1) Replace line 180 by:

```
180 LET A=USR 16514
```

2) Enter a line 10:

```
10 REM 123456
```

that reserves 6 locations for the machine code. The starting address being 16514.

3) Enter:

```
POKE 16514,42     (hex. 2A)
POKE 16515,12     (hex. 0C)
POKE 16516,64     (hex. 40)
POKE 16517,68     (hex. 44)
POKE 16518,77     (hex. 4D)
POKE 16519,201    (hex. C9)
```

which will enter a 6-byte machine code routine into line 10.

4) RUN the program.

The 'mnemonics' for this program are:

```
LD HL,(D-File)
LD B,H
LD C,L
RET
```

where as before the address of D-File has to be split into '12' and '64'.

The two new instructions 'LD B,H and LD C,L' are needed as the 8K ROM returns the value of the BC register pair rather than the HL register pair.

A Second Machine Code Routine

If you have followed the article so far, you might now like to try a longer machine code routine. Several new instructions will be introduced.

In the *Tower Game* the start of the Display File is used as a base address to which the variables L(R) and L(P) are added in turn. The resultant address then points to the location that is to be filled with a specific value. All of this procedure can be easily performed in machine code.

In the 4K program one proceeds as follows:

1) Replace lines 230 and 240 by:

```
230 POKE 16431,L(R)
```

which transfers the current value of L(R) to a suitable location for the machine code routine and

```
240 LET A=USR(16427)
```

which runs the machine code routine. Note how the variable A is just a dummy variable, i.e., not used later.

2) Replace line 270 by:

```
270 POKE 16441,L(P) and
275 LET A=USR(16437)
```

3) Replace line 10 by:

```
10 REM 12345678901234567890
```

4) And fill line 10 by using a loader such as:

```
500 FOR A=16427 TO 16446
510 INPUT B
520 POKE A,B
530 NEXT A
RUN 500
```

and enter the code:

42,12,64,17,0,0,25,54,0,201,42,12,64,17,0,0,25,54,9,201

5) Now delete the loader in lines 500-530 and RUN the whole program.

The machine code routine is given in Figure 4.

```
16427 LD   HL,(D-File)  Pick-up D-File.
      LD   DE,dddd      The offset.
      ADD  HL,DE        Form new
                        address.
      LD   (HL),+00     Blank out this
                        location.
      RET               Finished.
16437 LD   HL,(D-File)  Pick-Up D-File.
      LD   DE,dddd      The offset.
      ADD  HL,DE        Form new
                        address.
      LD   (HL),+09     Put a 'brick' in
                        this location.
      RET               Finished.
```

Figure 4.

The instruction 'LD DE,+dddd' loads a 2-byte constant into the DE register pair. In the routine the first byte is altered as required whereas the second byte always stays as zero. The instruction 'LD (HL),+dd' is used to load a constant into the location whose address is the current value of the HL register pair.

In the 8K program one proceeds in a similar manner.

1) Replace lines 10,180,190 and 220 by:

```
10 REM 12345678901234567890
180 POKE 16518,L(R)
190 LET A=USR 16514 or
190 RAND USR 16514 (which looks nice)
220 POKE 16528,L(P)
225 LET A=USR 16524 or
225 RAND USR 16524
```

2) Load line 10 by using:

```
500 FOR A=16514 TO 16533
510 INPUT B
520 POKE A,B
530 NEXT A
RUN 500
```

and enter:

42,12,64,17,0,0,25,54,0,201,42,12,64,17,0,0,25,54,8,201.

which is the same routine as given for the 4K version but with 'graphic 8' instead of 'graphic 9.'

3) Delete lines 500-530 and RUN the program.

The reader is now encouraged to try his own hand. For example, the variable C can be replaced entirely. This will, however, probably require the use the instructions in Figure 5.

A Bibliography

For those readers who wish to delve further into machine code, the following books are available (at least from U.K. suppliers).

Understanding Your ZX81 ROM by Ian Logan. £8.95. The Essential Software Company (Visconti Ltd.), 47, Brunswick Centre, London W1 CN 1AF, and other Melbourne House outlets. Need I say anything more than that this book deals extensively with the use of machine code in the ZX80/81 systems.

Mastering Machine Code on Your ZX81 or ZX80 by Tony Baker. £5.95. Interface, 44, Earls Court Road, London W8 6EJ. "Speak kindly of one's rivals and they will be kind to you."

Machine Language Programming for Your ZX80 & ZX81 £8.95. Melbourne House Publishers, 131, Trafalgar Rd., London SE 10.

The currently available books about the ZX81 are:

The ZX81 Campanion by Bob Maunder. £7.95 Linsac, 68, Barker Road, Linthorpe, Middlesborough Co. Cleveland TS5 5ES. A very good book. Deals more with 'computing theory' and less with the monitor than its predecessor.

The ZX81 Pocket Book by Trevor Toms. £4.95. Phipps Associates, 3, Downs Avenue, Epsom, Surrey KT 18 5HQ.

From *Interface*:

Getting Acquainted with your ZX81 by Tim Hartnell. £4.95.

30 Amazing Games for the ZX81 by Alistair Gourlay. £3.95.

50 Rip-Roaring Games for the ZX80 and ZX81. £4.95.

Hints & Tips for the ZX81. £4.25. Hewson Consultants, 7, Grahame Close, Blewbury, Oxon. OX 11 9QE.

Not Only 30 Programs for the Sinclair ZX81. £6.95. Essential Software Company (Visconti Ltd.), 47, Brunswick Centre, London W1 CN 1AF, and other Melbourne House outlets.

Again I would welcome seeing any programs written as a consequence of this article. ■

		dec.	hex.	
LD	A,+dd	62	3E	= LET A= ....
LD	A,(addr.)	58	3A	= LET A=PEEK.....
LD	(addr.),A	50	32	= POKE ... , A
INC	A	60	3C	= LET A=A+1

Figure 5.

4K ROM
1K RAM

Experiments in Memory and I/O Expansion

David G. Sommers

Introduction

I was pleasantly surprised to find that the Sinclair ZX80 has both the software and hardware facilities to expand memory and I/O with relative ease. This conclusion came after I had performed a few experiments on the ZX80.

I would like to share with you these experiments, their results, and my conclusions.

How the ZX80 Works

Before I could expand the memory or add I/O to a ZX80, I had to know how it worked.

The manual that comes with the ZX80 is of some help. There is also a hint into the dynamic nature of the operating system by virtue of the fact that system variables are in RAM. This allows them to be altered as needed by the operating system.

Though some may argue that the ZX80 is too simple to have an operating system, it does have a central program that allocates resources and passes control to the various subprograms resident in the machine.

The important resource we are concerned with here is memory, in particular the user memory.

The book *The ZX80 Companion*, published by Linsac, was most helpful in supplying other good information, albeit without an explanation of how RAMTOP knows how to get to the top of RAM.

RAMTOP is the physical top, or highest address of installed RAM. This top is marked by the top-of-stack (TOS) marker pair and 1095 (447h), the return address for all Basic line executions. See Figure 1.

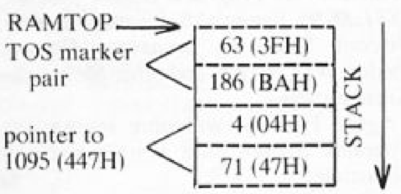


Figure 1.

The ZX80 has a very nice way of placing RAMTOP and these four special bytes at the top-of-stack.

David G. Sommers, 5155 Yarmouth Avenue, Apartment #9, Encino, CA 91316.

```
ADR  DT     New                    Comments

  0   21    LD HL, 7FFFh         ; TOP RAM ADDRESS
  1   FF
  2   7F
  3   3E    LD A, 3Fh            ; TERMINAL COUNT
  4   3F
  5   C3    JP 0261h             ; JUMP TO SEARCH FOR RAMTOP
  6   61
  7   02
  .    .
  .    .
261   36    LD(HL), 01h          ; STORE 01 IN ALL RAM
262   01
263   2B    DEC HL               ; NEXT ADDRESSS DOWN
264   BC    CP H                 ; HAS 4000h BEEN FILLED?
265   20    JR NZ, $-4           ; LOOP CONTROL
266   FA
267   23    INC HL               ; INC FROM 3FFFh
268   35   DEC(HL)               ; DEC VALUE IN RAM LOCATION HL
269   28   JR Z, $-2             ; LOOP CONTROL, IF 0,
                                   CONTINUE LOOPING
26A  FC
26B  F9    LD SP, HL             ; LOAD SP WITH RAMTOP
26C  F5    PUSH AF               ; START BUILDING TOP-OF-STACK
                                   MAKER
  .    .
  .    .
```

Figure 2.

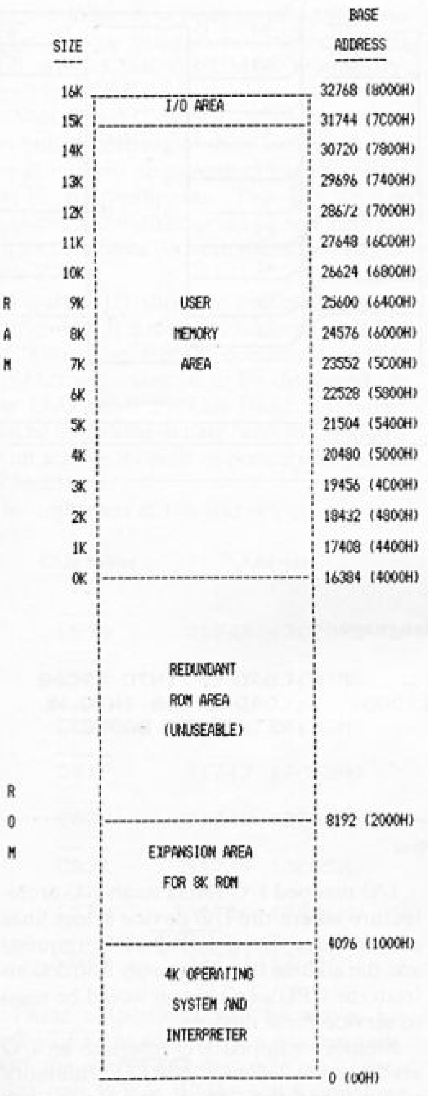
As part of the NEW command, which is executed on power-up or from the keyboard, the total amount of installed RAM is size checked by the subprogram I shall call NEW. See Figure 2. I disassembled NEW by hand using the PEEK(n) command to read the ROM. This routine starts by filling all possible 16K of RAM with 01h. Then, starting at location 16384 (4000h) (see Figure 3), each location will be decremented in value. Next, the Z flag in the CPU status register is checked to see if that location has become 00h in value. If it has, this is assumed a good memory location and the next location is tried. This continues until this test for zero fails. In the standard ZX80, this occurs at location 17408 (4400h). Here the actual location being addressed is 16384 (4000h). This is due to the simple linear-select for RAMCS (RAM chip select) explained in more detail later. What happens now is that this location, having been made 00h previously, now becomes FFh. This condition will cause the routine to exit its loop and load the stack pointer (SP) of the CPU with RAMTOP.

The search for RAMTOP has another interesting feature. During one experiment, that of adding 3K to the on-board 1K, I discovered I had only a total of 3K, and not the expected 4K. After many minutes of sweating and cursing the ZX80, I found that the very smart ZX80 had noticed that one of my added 2114 RAM chips was plugged in backwards, thus preventing the top 1K from reading back 00h during the search for RAMTOP. It is possible for a ZX80 to operate with bad memory chips as long as at least the first 1K or so still functions.

The last operational feature we need to know is how the chip selects are derived for RAM and ROM and how this knowledge might be applied to expansion.

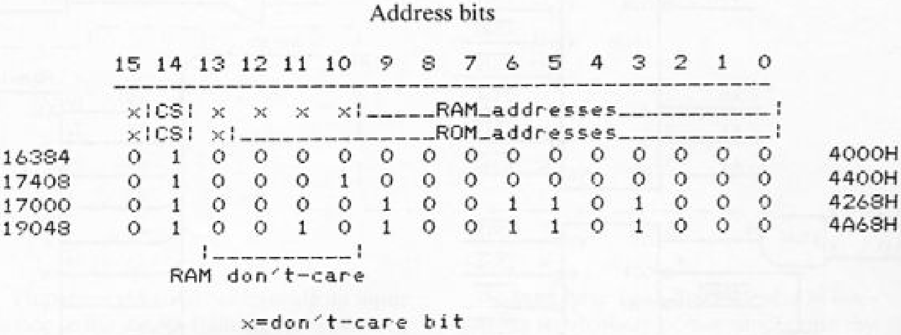
Consulting the ZX80 circuit diagram (*SYNC* 1:1, pp. 24-25) I noted that the ZX80 uses linear-selection for RAM and ROM. A14 (address bit 14) is used to divide the total address space in half. See Figure 3. When AA14 is true (high), RAM is selected; when false (low), ROM is selected. This works fine if there is only one ROM and one RAM (the pair of 2114 chips act as one 1K X 8 RAM).

The weakness of this linear-selection is that A10 through A13 are don't-care bits as far as RAM chip select is concerned. The ROM ignores A13 only. Figure 4 will help illustrate the problem.



```
                                              BASE
        SIZE                                ADDRESS
        ----                                -------

        16K ---------------------------- 32768 (8000H)
            :          I/O AREA         :
        15K :---------------------------: 31744 (7C00H)
        14K :                           : 30720 (7800H)
        13K :                           : 29696 (7400H)
        12K :                           : 28672 (7000H)
        11K :                           : 27648 (6C00H)
        10K :                           : 26624 (6800H)
R        9K :           USER            : 25600 (6400H)
A        8K :          MEMORY           : 24576 (6000H)
M        7K :           AREA            : 23552 (5C00H)
         6K :                           : 22528 (5800H)
         5K :                           : 21504 (5400H)
         4K :                           : 20480 (5000H)
         3K :                           : 19456 (4C00H)
         2K :                           : 18432 (4800H)
         1K :                           : 17408 (4400H)
         0K :---------------------------: 16384 (4000H)
            :                           :
            :                           :
            :         REDUNDANT         :
            :         ROM AREA          :
            :        (UNUSEABLE)        :
            :                           :
R           :                           :
O           :---------------------------: 8192 (2000H)
M           :      EXPANSION AREA       :
            :       FOR 8K ROM          :
            :                           :
            :---------------------------: 4096 (1000H)
            :       4K OPERATING        :
            :        SYSTEM AND         :
            :        INTERPRETER        :
            :                           :
            ----------------------------- 0 (00H)
```

Figure 3.



Address bits

```
          15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
          ----------------------------------------------
           x!CS! x  x  x  x!_____RAM_addresses__________!
           x!CS! x!_______________ROM_addresses__________!
16384      0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0      4000H
17408      0  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0      4400H
17000      0  1  0  0  0  0  1  0  0  1  1  0  1  0  0  0      4268H
19048      0  1  0  0  1  0  1  0  0  1  1  0  1  0  0  0      4A68H
                 !__________!
               RAM don't-care

                 x=don't-care bit
```

Figure 4.

If the ZX80 reads memory address 16384 (4000h), the address bus will have the bit pattern in Figure 4. The four don't-care bits (A10-A13) are zeroes, so there is no problem yet. Now the ZX80 tries to read 17408 (4400h). With A10 a don't-care bit, the RAM thinks I want 16384 again. The same thing happens if non-boundary addresses are tried. Figure 4 shows what occurs with 17000 (4268h) and 19048 (4A68h). Note address bit 11.

The ROM address space below 16384 (4000h) has a similar condition. If you are a sceptic, try this program:

```
10 FOR I=0 TO 4096
20 IF  NOT  PEEK(I)=PEEK(I+8192)
   THEN PRINT I;"AND";I+8192;
   "NOT EQUAL
30 NEXT I
```

This program will print any inequality between ROM addresses 0 through 4096 and the redundant ROM space from 8192 through 12284.

I will be using A10 through A13 to decode which 1K section of RAM I want.

Expansion into the ROM area will not be attempted here. It would require ANDing A14 with A13 into IC13 pin 13 to open up the address space from 8192 to 16383. I have considered doing this, along with adding an EPROM containing machine code peripheral drivers. The USR(n) command could then be used to access these routines.

Experiment in Hardware

I now know enough to design some expansion hardware. The first thing I need is a chip select decoder. I will use a 74LS138 1-of-8 decoder. See Figure 5-1. Then two 2114AL-4 RAM chips are added. This I quickly lash together on a small general purpose plug board.

The connector into the ZX80 was a little harder. It is necessary to make the connector by cutting down a standard .100" center P.C. card edge connector. A key for the slot at location 3 will also have to be fabricated.

I started with a 50 contact dual-readout connector for 1/16" board. The size of the connector I started with was immaterial. The larger versions of these connectors are not usually marked with the same alphanumeric numbering scheme as the ZX80 circuit diagram, making hookup a little harder.

I cut the connector with a hacksaw at the points illustrated in Figure 6. It is easier to cut the connector with the contacts at location 24 removed. After cleaning the cut areas with a file, I removed the two contacts at location 3 to make room for the key. On most connectors these contacts can be removed by pushing on them from the back until they start to move, then pull from the front. A pair of small needle-nose pliers will do the job. It is a good idea to save these removed contacts as spares. I managed to damage a needed contact and was thankful for my foresight. I made the key from a piece of 1/16" expoxy-glass unclad circuit board. I just snipped at it until it would force-fit in place. My ZX80 connector was now ready.

I was now able to complete the wiring of my first expansion board. I chose to ignore the fact that Sinclair connected data and address lines somewhat capriciously. You will notice that the ZX80 circuit diagram shows D7', D1', D0', and D2' going to IC3, and D4', D3', D5', and D6', going to IC4 in that order. I think pins 5 and 7 on IC3 and IC4 being crossed is a drafting error. Where is that famous British precision and desire for order? The reality is, of course, that it does not matter; it is only a convention anyway.

After a few visual checks.......the juice to it! I think so anyway. I quickly loaded a program from tape that was pushing the 1K RAM to the point of shrinking the display. I ran it.......no shrinking!

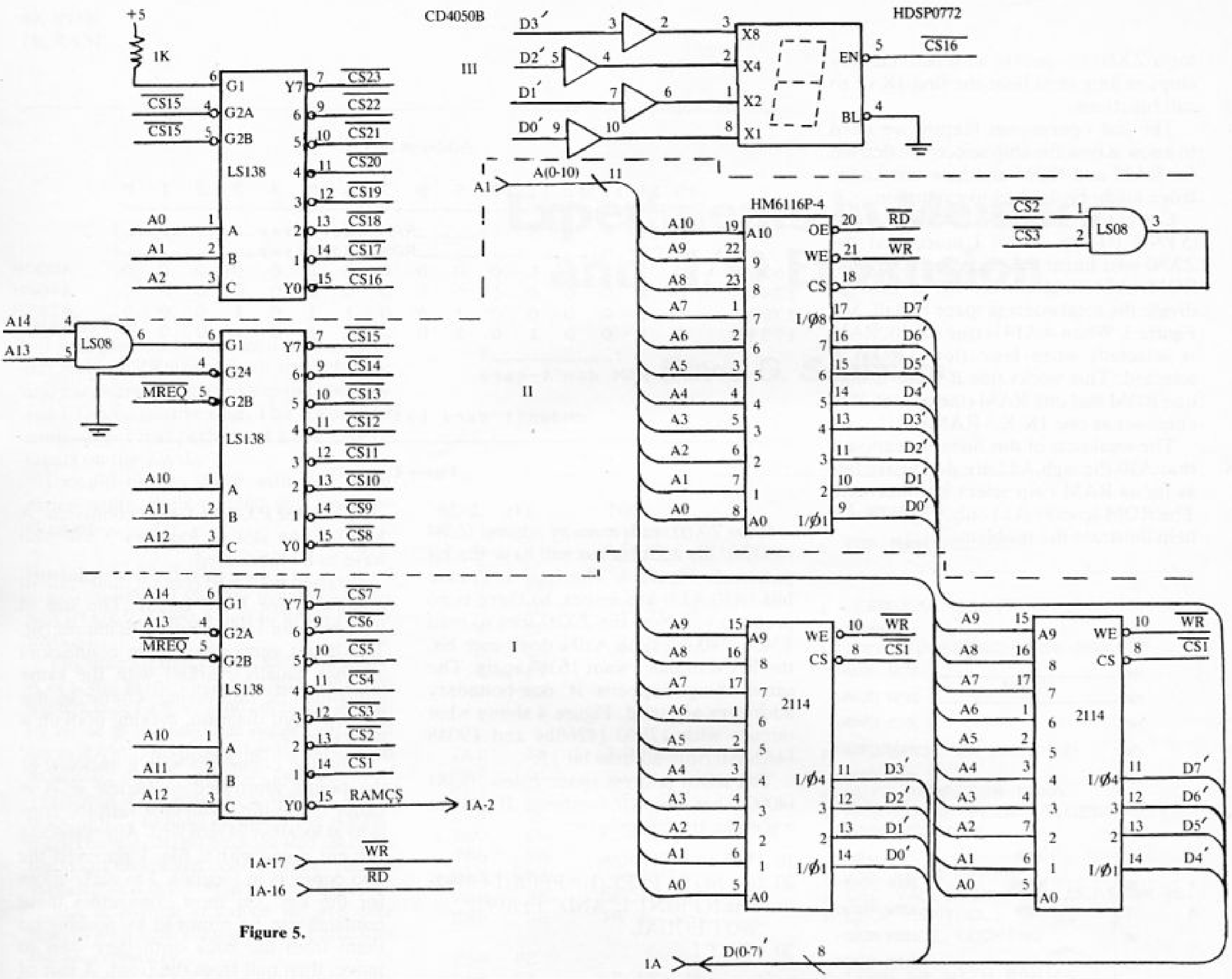


Figure 5.

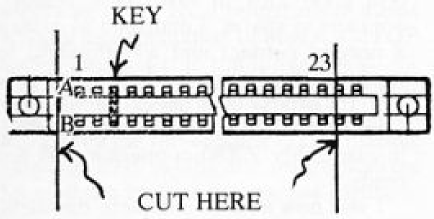


Figure 6.

I now need a little more definite check for the size of RAM. The program SIZER in Figure 7 will read the stack pointer, SP, in the CPU and print the number of bytes of RAM below it. When SIZER is run, the value returned to the screen is 14 bytes short of the total installed RAM. This is due to the ZX80 using the stack during execution of the program, thus the stack pointer is not pointing to the very top RAM.

My first experiment was a success. I now have the confidence to explore further.

In Basic

```
100 LET P=17000
110 POKE P,237
120 POKE P+1,115
130 POKE P+2,112
140 POKE P+3,66
150 POKE P+4,42
160 POKE P+5,112
170 POKE P+6,66
180 POKE P+7,201
190 LET SP=USR(P)
200 PRINT ,SP-16384;"#BYTES"
```

In assembly language:

```
LD (17008),    SP     ;LOAD SP INTO 17008
LD HL,    (17008)     ;LOAD 17008 INTO HL
RET                   ;RETURN TO BASIC
```

Figure 7. Sizer

I/O Expansion

By now I/O expansion seems trivial; however, it is not totally trivial. I decided that using I/O mapped I/O would be too tricky due to the way the I/O address space is consumed with the keyboard. Besides, the PEEK (n) and POKE n,n commands operate in memory address space. So memory mapped I/O will be the way.

I/O mapped I/O refers to an I/O architecture where the I/O device select lines are decoded from IORQ (I/O request) and the address bus. Thus, only instructions from the CPU's I/O group would be used to service these devices.

Memory mapped I/O refers to an I/O architecture where MREQ (memory request) and the address bus are used to decode the device selects. Here, the device appears as one or more memory locations to the CPU, and is treated such. The full array of memory instructions can be used on these I/O devices. In the ZX80, PEEK (n) and POKE n,n are available.

One disadvantage with memory mapped I/O in general is that it consumes part of the memory address space. This is 1K in my design for the ZX80.

My first I/O device will be simple enough. I have some Hewlett Packard displays; HDSP0772 to be exact. These hexadecimal LED displays have a built-in latch memory, decoder driver, and run from +5 VDC. They also can be spoken to as if they were a memory chip with one address. The only trick I need to play is to buffer the data bus of the ZX80. The group of resistors, R4 through R11, in the ZX80, that presumably are in series with the outgoing data lines to prevent a bus crash, render the data bus rather puny. I will use a CD4050B CMOS hex buffer. See Figure 5-III.

NMOS and CMOS RAM chips require no buffer because of their extremely low input current requirement on the order of 10 microamperes. The HP display requires 1.6 milliamperes of sink current on its data lines—a veritable short circuit to a ZX80.

Figure 5-III shows a buffered display configured at a memory address of 31744 (7C00h). The simple command POKE 31744,n will cause n to be displayed on the LED readout. This Basic statement can be imbedded in long running programs as an activity monitor or possibly for games as feedback.

The addresses of the I/O selects are:

Chip Select	Address
$\overline{\text{CS16}}$	31744 (7C00H)
$\overline{\text{CS17}}$	31745 (7C01H)
$\overline{\text{CS18}}$	31746 (7C02H)
$\overline{\text{CS19}}$	31747 (7C03H)
$\overline{\text{CS20}}$	31748 (7C04H)
$\overline{\text{CS21}}$	31749 (7C05H)
$\overline{\text{CS22}}$	31750 (7C06H)
$\overline{\text{CS23}}$	31751 (7C07H)

These chip-selects will be active for both PEEK (n) and POKE n,n commands. With the simple I/O device in my experiment, only POKE 31744,n will do anything noticeable.

I have not yet tried configuring an input device to the ZX80. I anticipate no serious problems in using a tri-state bus driver, such as a 74LS244, connected to CS17 and the unbuffered data bus. This is to be used to talk to the ZX80 from switches or other devices. I simply ran out of board space on my little plug board.

Further Memory Expansion

I have come into possession of a pair of Hitachi HM6116P-4 CMOS RAM chips. These 2K X 8 monsters are ideal for further expansion of the ZX80 memory. They are full static; no clocks or strobes. Even the slowest family member, the -4, is more than fast enough at 200 nanoseconds maximum access time. What is most desirable is their low power consumption: 100 microwatts deselected and 180 milliwatts operating. By contrast, the ubiquitous 2114AL-4 consumes 125 milliwatts, selected or not, and has 1/4 the bit capacity. The beauty of the Hitachi CMOS grows pale when the price and availablity of these exotic parts are considered. I was lucky to get two of them cheaply.

The chip-select scheme in Figure 5-I and 5-II allows for a mix of 1K X 4 and 2K X 8 parts. By using as many chip-select lines as I have chips for, I can configure any size memory I want. Please note in Figure 5-II that two chip-select lines are required for each 2K X 8 RAM and that R19 in the ZX80 allows $\overline{\text{CSO}}$ to override the chip select from IC12 pin 8. At present I have 6K of memory operating; the 1K inside, two HM6116P-4, and two 2114AL-4.

Consideration of Power and Loading

Power consumption and circuit loading are very important in general, and crucial in a ZX80.

I have estimated that the reserve current in the ZX80 power system is about 100 milliamperes. At this level the wall unit and the ZX80 itself gets toasty warm. Even this moderate heating will cause some strange actions to occur. I notice the LOAD function failed to operate after the ZX80 had cooked with a 100 milliampere load.

So far I have been unsuccessful in integrating an auxiliary power supply into my expansion system. My problem seems to be 60 Hertz line noise getting into the video. Most annoying. I believe my problem stems from the fact that I am using a TV set with a hot chassis and a slightly leaky power transformer in the auxiliary power supply I was attempting to use. I will fight this problem some other time.

Though auxiliary power would solve the power problem, so would the careful selection of parts. The 74LS138 1-of-8 decoder is fairly stingy at 6.3 milliamperes. The Intel equivalent 8205 is an outrageous hog at 70 milliamperes worst case, possibly 35 milliamperes typical. One must be cautious of functionally equivalent parts. Look hard at the specifications.

CMOS will solve the power and loading problems. The only problem with CMOS, especially RAMs, is cost and availability. Time will cure this, but I want it now.

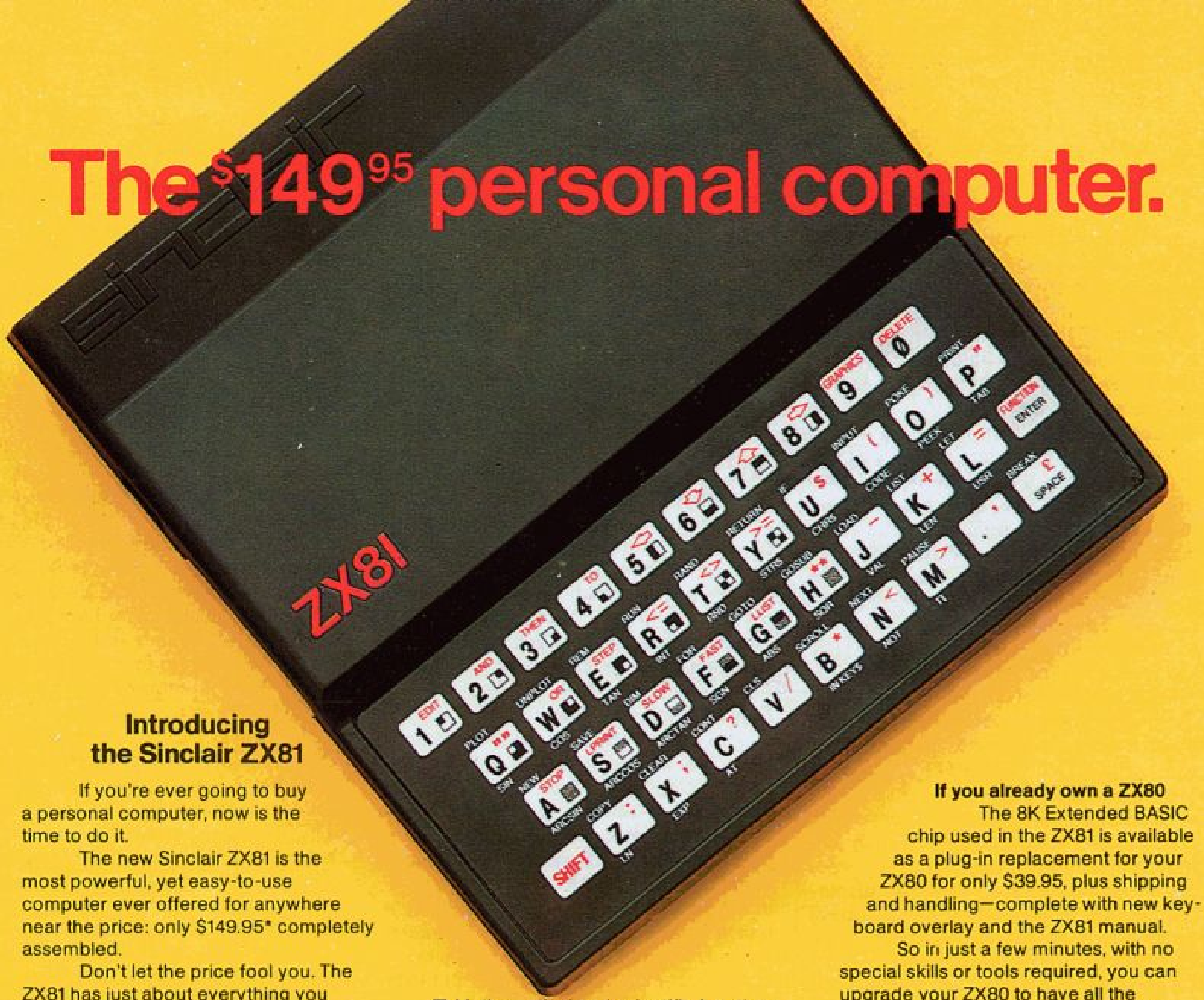
That beautiful HP LED display I used is a terrible glutton for power. That particular display is only good for demonstration of the ZX80 output capability. There is a low-power version of this display available.

Circuit loading is most crucial with I/O expansion. As explained earlier, the ZX80 expansion data bus is quite puny. This requires careful reading of chip specifications for input loading. A 74LS244 part has a special PNP input circuit that loads its source much less than standard low-power Schottky. This allows direct connection to the ZX80 data bus without a CMOS buffer. Though workable, this is still marginal at best. More exotic parts, such as the 8212 I/O port, have even less loading on their inputs. Again, the 8212, though versatile, consumes 90 milliamperes typically.

Conclusions

The ZX80 is definitely expandable with ease. The operating system has been designed with memory expansion in mind. With the compromises that life presents us daily, a very powerful version of the ZX80 can be configured at moderate cost.

The $149^{95} personal computer.



Introducing the Sinclair ZX81

If you're ever going to buy a personal computer, now is the time to do it.

The new Sinclair ZX81 is the most powerful, yet easy-to-use computer ever offered for anywhere near the price: only $149.95* completely assembled.

Don't let the price fool you. The ZX81 has just about everything you could ask for in a personal computer.

A breakthrough in personal computers

The ZX81 is a major advance over the original Sinclair ZX80—the world's largest selling personal computer and the first for under $200.

In fact, the ZX81's new 8K Extended BASIC offers features found only on computers costing two or three times as much.

Just look at what you get:

- Continuous display, including moving graphics
- Multi-dimensional string and numerical arrays
- Mathematical and scientific functions accurate to 8 decimal places
- Unique one-touch entry of key words like PRINT, RUN and LIST
- Automatic syntax error detection and easy editing
- Randomize function useful for both games and serious applications
- Built-in interface for ZX Printer
- 1K of memory expandable to 16K

The ZX81 is also very convenient to use. It hooks up to any television set to produce a clear 32-column by 24-line display. And you can use a regular cassette recorder to store and recall programs by name.

If you already own a ZX80

The 8K Extended BASIC chip used in the ZX81 is available as a plug-in replacement for your ZX80 for only $39.95, plus shipping and handling—complete with new keyboard overlay and the ZX81 manual.

So in just a few minutes, with no special skills or tools required, you can upgrade your ZX80 to have all the powerful features of the ZX81. (You'll have everything except continuous display, but you can still use the PAUSE and SCROLL commands to get moving graphics.)

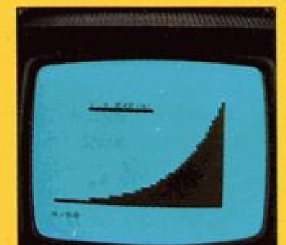
With the 8K BASIC chip, your ZX80 will also be equipped to use the ZX Printer and Sinclair software.

Warranty and Service Program**

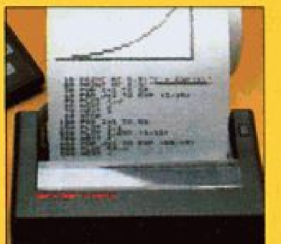
The Sinclair ZX81 is covered by a 10-day money-back guarantee and a limited 90-day warranty that includes free parts and labor through our national service-by-mail facilities.

*Plus shipping and handling. Price includes connectors for TV and cassette, AC adaptor, and FREE manual.

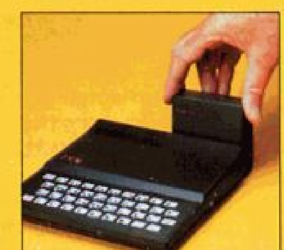
**Does not apply to ZX81 kits.



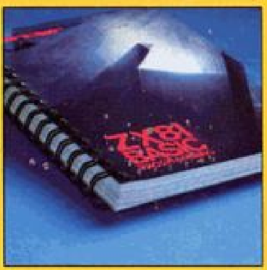
NEW SOFTWARE: Sinclair has published pre-recorded programs on cassettes for your ZX81, or ZX80 with 8K BASIC. We're constantly coming out with new programs, so we'll send you our latest software catalog with your computer.



ZX PRINTER: The Sinclair ZX Printer will work with your ZX81, or ZX80 with 8K BASIC. It will be available in the near future and will cost less than $100.



16K MEMORY MODULE: Like any powerful, full fledged computer, the ZX81 is expandable. Sinclair's 16K memory module plugs right onto the back of your ZX81 (or ZX80, with or without 8K BASIC). Cost is $99.95, plus shipping and handling.



ZX81 MANUAL: The ZX81 comes with a comprehensive 164-page programming guide and operating manual designed for both beginners and experienced computer users. A $10.95 value, it's yours free with the ZX81.

The $^{\$}99^{95}$ personal computer.

Introducing the ZX81 kit

If you really want to save money, and you enjoy building electronic kits, you can order the ZX81 in kit form for the incredible price of just $99.95* It's the same, full-featured computer, only you put it together yourself. We'll send complete, easy-to-follow instructions on how you can assemble your ZX81 in just a few hours. All you have to supply is the soldering iron.

How to order

Sinclair Research is the world's largest manufacturer of personal computers.

The ZX81 represents the latest technology in microelectronics, and it picks up right where the ZX80 left off. Thousands are selling every week.

We urge you to place your order for the new ZX81 today. The sooner you order, the sooner you can start enjoying your own computer.

To order, simply call our toll free number, and use your MasterCard or VISA.

To order by mail, please use the coupon. And send your check or money order. We regret that we cannot accept purchase orders or C.O.D.'s.

CALL 800-543-3000. Ask for operator #509. In Ohio call 800-582-1364. In Canada call 513-729-4300. Ask for operator #509. Phones open 24 hours a day, 7 days a week. Have your MasterCard or VISA ready.

These numbers are for orders only. For information, you must write to Sinclair Research Ltd., One Sinclair Plaza, Nashua, NH 03061.

sinclair

AD CODE	PRICE†	QTY.	AMOUNT
ZX81	$149.95		
ZX81 Kit	99.95		
8K BASIC chip (for ZX80)	39.95		
16K Memory Module (for ZX81 or ZX80)	99.95		
Shipping and Handling	4.95		$4.95
To ship outside USA add $10.00			
		TOTAL	

MAIL TO: Sinclair Research Ltd., One Sinclair Plaza, Nashua, NH 03061.

NAME____________________

ADDRESS____________________

CITY/STATE/ZIP____________________

† U.S. Dollars

8K ROM
16K RAM

An Inventory System

Dr. Stephen A. Justham

Mass data storage is accomplished more efficiently by a disc system than by a cassette recorder. However, until such a system is available for the ZX80/1 computers, Sinclair owners will have to rely on the cassette system. This article offers a program for a modest inventory system based on the 8K ROM and 16K RAM.

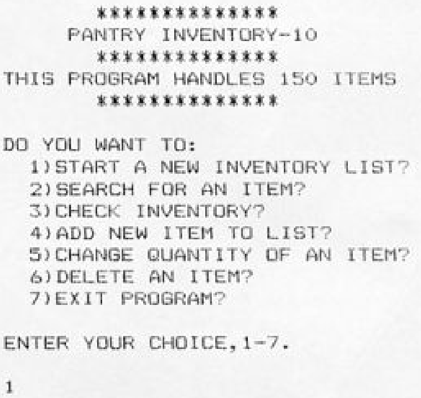
For the sake of illustration, a "pantry" inventory is used, but the program can be adapted to any inventory you might want to use it for. The program will handle up to 150 separate items (Figure 1), but it can be easily modified for the individual user (lines 305, 307, 3044, and all the "B=1 TO 150" statements).

One begins by selecting choice 1, indicating the total number of items to be entered (Figure 2), and INPUTing the item name and quantity in response to prompts (Figures 3 and 4). Once an inventory list is entered, the user has several options.

The search routine accepts a string input (Figure 5) and searches the inventory to ascertain if the item is in the listing. After searching, the computer replies with either a report on the item it has located (Figure 6) or a statement indicating that the item is not in the inventory (Figure 7). A complete inventory listing is available with the third option. The program lists each entry by item number, name, and quantity (Figure 8). If the list is long, there is an option at the end of the program to re-check the listing.

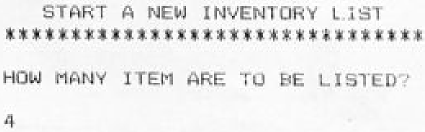
New items are added to the inventory by the fourth option. The current inventory is listed and a prompt requests the new item name and quantity (Figure 9). When

Dr. Stephen A. Justham, 8300 N. Costa Mesa Dr., Muncie, IN 47302.



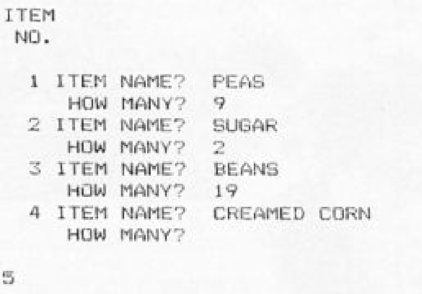
```
      **************
    PANTRY INVENTORY-10
      **************
THIS PROGRAM HANDLES 150 ITEMS
      **************

DO YOU WANT TO:
  1)START A NEW INVENTORY LIST?
  2)SEARCH FOR AN ITEM?
  3)CHECK INVENTORY?
  4)ADD NEW ITEM TO LIST?
  5)CHANGE QUANTITY OF AN ITEM?
  6)DELETE AN ITEM?
  7)EXIT PROGRAM?

ENTER YOUR CHOICE,1-7.

1
```

Figure 1.



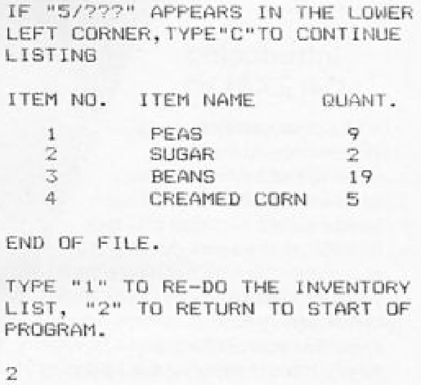
```
   START A NEW INVENTORY LIST
********************************

HOW MANY ITEM ARE TO BE LISTED?

4
```

Figure 2.



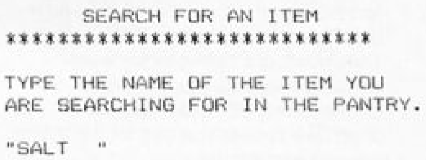
```
ITEM
 NO.

  1 ITEM NAME?  PEAS
     HOW MANY?  9
  2 ITEM NAME?  SUGAR
     HOW MANY?  2
  3 ITEM NAME?  BEANS
     HOW MANY?  19
  4 ITEM NAME?  CREAMED CORN
     HOW MANY?

5
```

Figure 3.



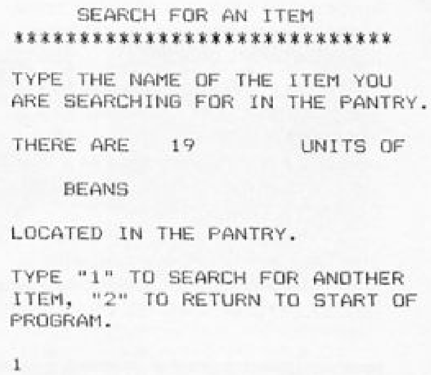
```
IF "5/???" APPEARS IN THE LOWER
LEFT CORNER,TYPE"C"TO CONTINUE
LISTING

ITEM NO.  ITEM NAME      QUANT.

   1        PEAS           9
   2        SUGAR          2
   3        BEANS          19
   4        CREAMED CORN   5

END OF FILE.

TYPE "1" TO RE-DO THE INVENTORY
LIST, "2" TO RETURN TO START OF
PROGRAM.

2
```

Figure 4.



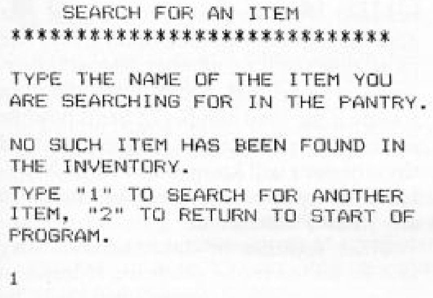
```
      SEARCH FOR AN ITEM
****************************

TYPE THE NAME OF THE ITEM YOU
ARE SEARCHING FOR IN THE PANTRY.

"SALT  "
```

Figure 5.



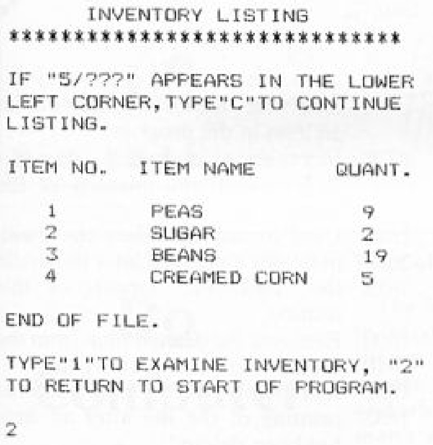
```
      SEARCH FOR AN ITEM
*****************************

TYPE THE NAME OF THE ITEM YOU
ARE SEARCHING FOR IN THE PANTRY.

THERE ARE   19        UNITS OF

    BEANS

LOCATED IN THE PANTRY.

TYPE "1" TO SEARCH FOR ANOTHER
ITEM, "2" TO RETURN TO START OF
PROGRAM.

1
```

Figure 6.



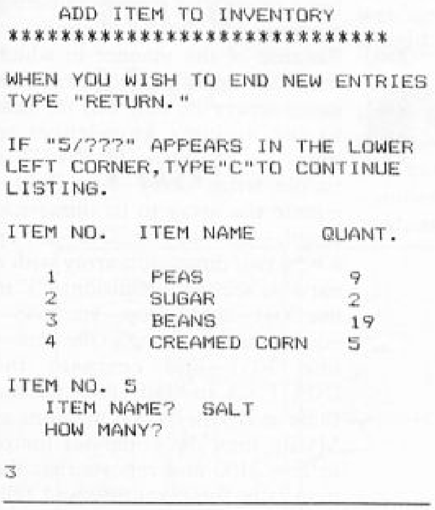
```
    SEARCH FOR AN ITEM
******************************

TYPE THE NAME OF THE ITEM YOU
ARE SEARCHING FOR IN THE PANTRY.

NO SUCH ITEM HAS BEEN FOUND IN
THE INVENTORY.
TYPE "1" TO SEARCH FOR ANOTHER
ITEM, "2" TO RETURN TO START OF
PROGRAM.

1
```

Figure 7.



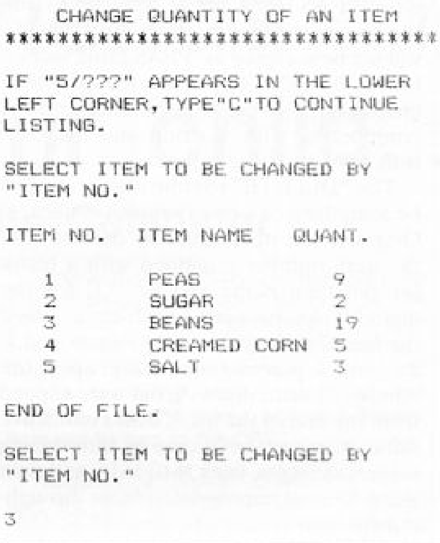
```
      INVENTORY LISTING
******************************

IF "5/???" APPEARS IN THE LOWER
LEFT CORNER,TYPE"C"TO CONTINUE
LISTING.

ITEM NO.   ITEM NAME       QUANT.

    1        PEAS            9
    2        SUGAR           2
    3        BEANS           19
    4        CREAMED CORN    5

END OF FILE.

TYPE"1"TO EXAMINE INVENTORY, "2"
TO RETURN TO START OF PROGRAM.

2
```

Figure 8.



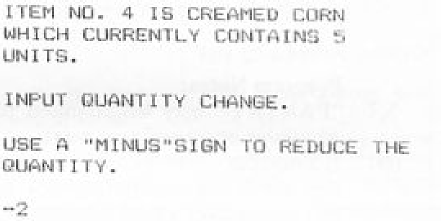
```
    ADD ITEM TO INVENTORY
*****************************

WHEN YOU WISH TO END NEW ENTRIES
TYPE "RETURN."

IF "5/???" APPEARS IN THE LOWER
LEFT CORNER,TYPE"C"TO CONTINUE
LISTING.

ITEM NO.   ITEM NAME      QUANT.

    1        PEAS           9
    2        SUGAR          2
    3        BEANS          19
    4        CREAMED CORN   5

ITEM NO. 5
   ITEM NAME?  SALT
   HOW MANY?

3
```

Figure 9.



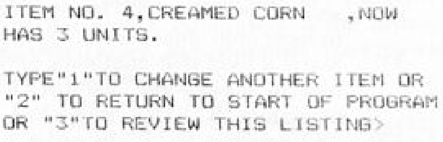
```
    CHANGE QUANTITY OF AN ITEM
*********************************

IF "5/???" APPEARS IN THE LOWER
LEFT CORNER,TYPE"C"TO CONTINUE
LISTING.

SELECT ITEM TO BE CHANGED BY
"ITEM NO."

ITEM NO.  ITEM NAME   QUANT.

    1        PEAS           9
    2        SUGAR          2
    3        BEANS          19
    4        CREAMED CORN   5
    5        SALT           3

END OF FILE.

SELECT ITEM TO BE CHANGED BY
"ITEM NO."

3
```

Figure 10.



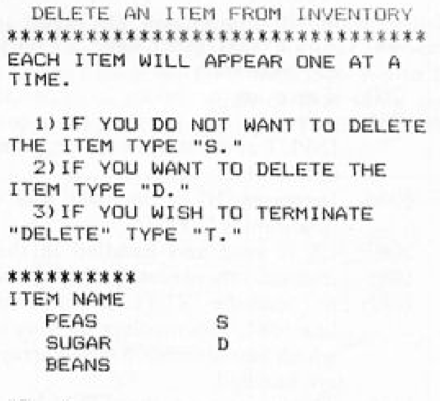
```
ITEM NO. 4 IS CREAMED CORN
WHICH CURRENTLY CONTAINS 5
UNITS.

INPUT QUANTITY CHANGE.

USE A "MINUS"SIGN TO REDUCE THE
QUANTITY.

-2
```

Figure 11.



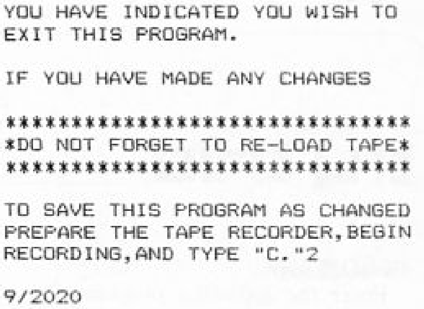
```
ITEM NO. 4,CREAMED CORN    ,NOW
HAS 3 UNITS.

TYPE"1"TO CHANGE ANOTHER ITEM OR
"2" TO RETURN TO START OF PROGRAM
OR "3"TO REVIEW THIS LISTING>
```

Figure 12.



```
  DELETE AN ITEM FROM INVENTORY
*********************************
EACH ITEM WILL APPEAR ONE AT A
TIME.

  1)IF YOU DO NOT WANT TO DELETE
THE ITEM TYPE "S."
  2)IF YOU WANT TO DELETE THE
ITEM TYPE "D."
  3)IF YOU WISH TO TERMINATE
"DELETE" TYPE "T."

**********
ITEM NAME
   PEAS          S
   SUGAR         D
   BEANS

"T  "
```

Figure 13.



```
YOU HAVE INDICATED YOU WISH TO
EXIT THIS PROGRAM.

IF YOU HAVE MADE ANY CHANGES

******************************
*DO NOT FORGET TO RE-LOAD TAPE*
******************************

TO SAVE THIS PROGRAM AS CHANGED
PREPARE THE TAPE RECORDER,BEGIN
RECORDING,AND TYPE "C."2

9/2020
```

Figure 14.

the addition of items is completed "RETURN" is typed to send the program back to the menu at the beginning of the program.

Since an inventory must always accommodate changes in the quantity of the items, the quantity change routine is very important. Option five, which first prints the complete inventory, asks for the item number of the item to be changed (Figure 10). After the item is selected, it is detailed and the change is requested (Figure 11). Finally, the item number, name, and quantity change are printed out. The user can then return to the start of the routine to change another item (Figure 12).

The last option, aside from exiting the program, permits the deletion of an item from the inventory list. An item is printed and the user inputs one of three choices: save, delete, or terminate the routine (Figure 13).

Exiting from the program and SAVEing the program are combined in the last option. The prompts remind the user to re-load the tape with the up-dated inventory list (Figure 14).

Several minor problems were encountered in attempting to develop a workable inventory-type program for the ZX80/1. Most notable of the "minor" problems involved the way the 8K ROM handles string arrays. This problem can to light when the "SEARCH" routine was first (and second, and third, and ...) attempted. Finally, the attempt to use a "SEARCH" routine was set aside. The solution to the problem, which involved INPUTing an "ITEM NAME," having the inventory checked by the computer and then reporting whether or not the item appears in the inventory came from a technique used in another part of the program, lines 1080-1084.

The difficulty involved in using the two-dimension array is in the second dimension. Once set (or simply using the 8K Basic ROM's own setting of 10 characters in length) the ZX80 only recognizes an item with the same number of second dimension characters. For example, if an array statement reads DIM (5,5), then five items, five characters in length may be input. If "PEAS" is typed in as an inventory item the computer will store it as "PEAS blank space." In the search routine one must

try this

4K ROM users:

Enter the following program (this is especially for ZX80 users in the US; other national users are invited to contribute):

```
   10 LET A$=" |ENTER
5 SHIFT H AND 15
SHIFT W| "
   20 LET B$=" |ENTER
25 SHIFT W| "
   30 FOR B=1 TO 5
   40 PRINT A$
   50 NEXT B
   60 FOR B=1 TO 7
   70 PRINT B$
   80 NEXT B
```

Our thanks to:

Ted A. Pozyski
1215 Thompson St.
Houston, TX 77007

8K ROM users:

Enter the following program:

```
   10 LET L=USR 2193
   20 FOR A=1 TO 100
   30 NEXT A
   40 GOTO 10
```

RUN the program. When the cursor reappears enter the program again and RUN. (ZX81: be sure to use fast mode.) Now turn up the volume on your TV set to give the best output. You can use the BREAK key to escape from the loop. This one might even have a practical application.

Our thanks to:

L. Richardson
82 Elgar Cres.
Cardiff, CF3 9RW
United Kingdom

type "PEAS blank space" in order to find the item in the inventory. Typing "PEAS" will not be accepted as "PEAS blank space." Lines 3040-3068 overcome this problem, mainly through the LEN function used in conjunction with a string and the one-time use of a string array.

The "DELETE" routine also proved to be something of a programming challenge. Originally an item could be deleted, but the item number remained with a blank for the item name and an "O" for the quantity. An associated problem involved the fact that the items were not moved 1, 2,..., or n places (depending upon the number of items deleted), but were clipped from the end of the list. These cumulative difficulties were overcome by introducing several variables, lines 1610-1618, and then using them at appropriate places throughout the routine; notably lines 1820, 1840, and 1908. Other manipulations such as found in lines 1830, 1835, 1900, 1909, and 1920 were employed to achieve the desired deletion and renumbering results.

Many "frills" have been used in the program and may be easily eliminated if so desired. For instance, the asterisks and many PRINT statements may be removed without affecting the program. These frill lines are marked by an * in the program. The total number of items may also be reduced. (NOTE: be sure to keep the first dimension of the two-dimension arrays one more than the total number desired, otherwise problems may occur in the deletion routine if the total number of items possible is used.) In all cases, it has been assumed that the user will know when to hit NEWLINE, therefore, this does not appear in any PRINT statement.

Other routines or data variables may also be added to the program. If location is important this variable may easily be included possibly as a string variable. More generally, this program may be readily adapted to any type of inventory situation.

Program Notes:

5,7 "PANTRY" may be changed to whatever inventory you want.

160 If a number other than 1-7 is typed, this sends the computer back to 90 to start over again.

305 Sets the first dimension of the two dimension array equal to one more than necessary for the "DELETE" routine to function properly. The second dimension may be changed to meet individual needs.

332 Starts a loop that continues until told to leave—line 390—or the maximum "B"—150—is reached.

370 "X" equated to "B" in order to
375 evaluate "N" without involving "B" directly.

400 This and similar lines may be removed if the prompt is not needed.

490 Any INPUT other than "P" starts the program over.

960 These are necessary in order to
965 increase the number of items INPUT at the start of the program in line 310.

1040 Increases "B" by one each time a new item is INPUT.

1080 C$ is used and handled in this
1081 manner; otherwise the ZX80/1 will
1082 not read the "RETURN" order in line 1084. This involves the way in which two dimension string arrays are handled.

1125 This line removes the "ITEM NO.," "ITEM NAME," and "QUANT." line on which "RETURN" is typed, otherwise "RETURN" shows up as an item in the program.

1370 Increases or decreases (if minus sign is used) the quantity of the item.

1610 Used to accommodate the mani-
1615 pulations employed later to handle
1618 the "DELETE" aspects of this routine.

1800 Removes the deleted item from the
1810 file then sets the quantity to 0.

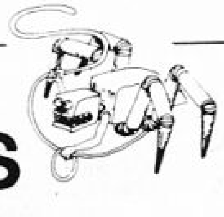
1830 Increments "B" by one to continue
1835 printing of the file after an item has been deleted.

1900 Decrements "N" by one for each item deleted.

1908 Re-sets "B" equal to what it had been originally, for renumbering purposes following a deletion.

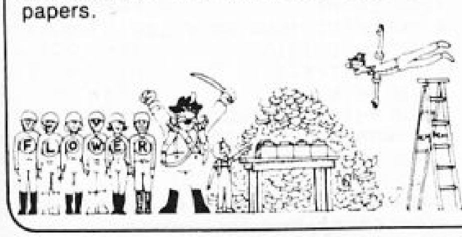
3040 Because of the manner in which
3042 the ZX80 handles two dimension
3044 string arrays the only way (at least
3048 to the author's knowledge) to
3052 initiate a search is to INPUT a simple string array—line 3040—equate the array to its numerical length—LEN in line 3042—set up a new two dimension array with a variable second dimension—"J" in line 3044—start a loop—line 3048—equate the new string to the item—line 3052—and compare the INPUT, C$, to M$(B) (which is the same as I$(B)). If C$ is the same as M$(B), then the computer jumps to line 3100 and reports that the item is in the inventory and tells how many units are present.

new friends for your child...



Katie and the Computer

Fred D'Ignazio and Stan Gilliam have created a delightful picture book adventure that explains how a computer works to a child. Katie "falls" into the imaginary land of Cybernia inside her Daddy's home computer. Her journey parallels the path of a simple command through the stages of processing in a computer, thus explaining the fundamentals of computer operation to 4 to 10 year olds. Supplemental explanatory information on computers, bytes, hardware and software is contained in the front and back end papers.



Thrill with your chidren as they join the Flower Bytes on a bobsled race to the CPU. Share Katie's excitement as she encounters the multi-legged and mean Bug who lassoes her plane and spins her into a terrifying loop. Laugh at the madcap race she takes with the Flower Painters by bus to the CRT.

"Towards a higher goal, the book teaches the rewards of absorbing the carefully-written word and anticipating the next page with enthusiasm..."

The Leader

"Children might not suspect at first there's a method to all this madness—a lesson about how computers work. It does its job well."

The Charlotte Observer

"...the book is both entertaining and educational."

Infosystems

Order Today

Katie and the Computer is hardbound, illustrated in full color throughout and costs just $6.95. A T-Shirt picturing the program bug in the story is also available (purple bug on a beige shirt). Shirts are available in adult S, M, L, XL, children's S, M and L and cost just $6.00.

To order send payment plus $2.00 postage and handling per order to the address below.

creative computing

39 E. Hanover Avenue
Morris Plains, NJ 07950
Toll-free **800-631-8112**
In NJ 201-540-0445

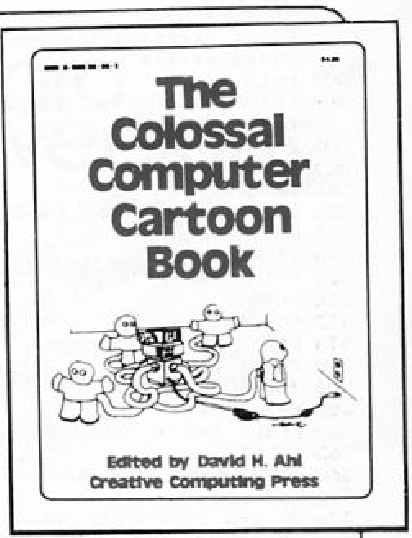
Do Computer Enthusiasts Have More Fun?

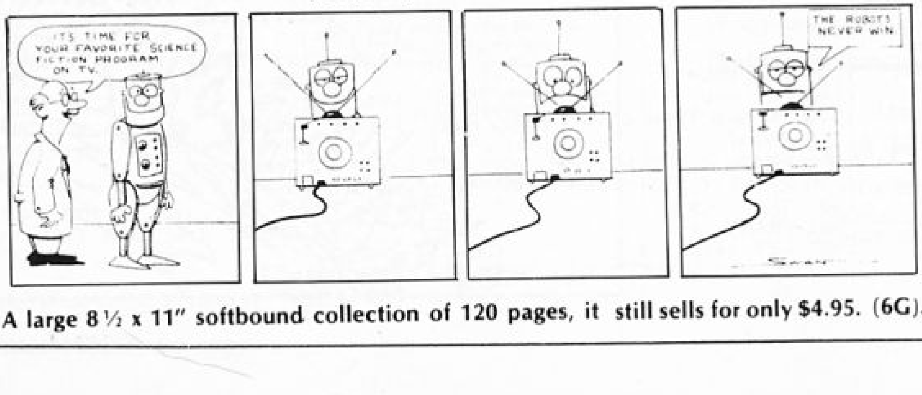
The Colossal Computer Cartoon Book

The best collection of computer cartoons ever is now in its second printing, and sports a bright new cover. The fifteen chapters contain hundreds of cartoons about robots, computer dating, computers in the office, home, and lab, and much more. 36 cartoonists share their views of man's ultimate machine.

Keep this book with your reference works. When needed, the right cartoon can say it all for you. When you need a break from debugging a good laugh can give you a welcome lift. Recommended for hours of fun and comic insight.

Edited by David Ahl, mastermind behind the April Fool's issue of Dr. Kilobyte's Creative Popular Personal Recreational Micro Computer Data Interface World Journal, this cartoon book contains much of that same incurable zaniness. [Want this issue? It's April 1980 and only $2.50 postpaid].





To order, send $4.95 plus $2.00 shipping and handling to Creative Computing Press, P.O. Box 789-M, Morristown, N.J. Or call in your MasterCard, Visa, or American Express order toll free: 800-631-8112 (N.J.(201)-540-0445).

A large 8½ x 11" softbound collection of 120 pages, it still sells for only $4.95. (6G).

Listing:

```
   2 REM STEPHEN A.> JUSTHAM,.8-5-81
   5 REM "PANTRY INVENTORY-10"
  *6 PRINT TAB 7;"***************
     "
   7 PRINT TAB 5;"PANTRY INVENTO
     RY-10"
  *9 PRINT TAB 7;"***************
     "
  10 PRINT "THIS PROGRAM HANDLES
     150 ITEMS"
 *11 PRINT TAB 7;"***************
     "
  15 PRINT
  20 PRINT "DO YOU WANT TO:"
 *25 PRINT
  30 PRINT TAB 2;"1)START A NEW
     INVENTORY LIST?"
  40 PRINT TAB 2;"2)SEARCH FOR A
     N ITEM?"
  65 PRINT TAB 2;"3)CHECK INVENT
     ORY?"
  70 PRINT TAB 2;"4)ADD NEW ITEM
     TO LIST?"
  75 PRINT TAB 2;"5)CHANGE QUANT
     ITY OF AN ITEM?"
  80 PRINT TAB 2;"6)DELETE AN IT
     EM?"
  86 PRINT TAB 2;"7)EXIT PROGRAM
     ?"
 *90 PRINT
  92 PRINT "ENTER YOUR CHOICE,1-
     7."
  96 INPUT A
  99 CLS
*100 IF A=1 THEN GOTO 300
 110 IF A=2 THEN GOTO 3000
 120 IF A=3 THEN GOTO 600
 130 IF A=4 THEN GOTO 900
 140 IF A=5 THEN GOTO 1200
 150 IF A÷6 THEN GOTO 1500
 155 IF A=7 THEN GOTO 2000
 160 GOTO 90
 300 PRINT TAB 3;"START A NEW IN
     VENTORY LIST"
*301 PRINT "********************
     ************"
 302 PRINT
 304 PRINT "HOW MANY ITEMS ARE T
     O BE LISTED?"
 305 DIM I$(151,15)
 307 DIM Q(151)
 310 INPUT N
 315 CLS
 322 PRINT
 324 PRINT "ITEM"
 326 PRINT TAB 1;"NO."
 328 PRINT
 332 FOR B=1 TO 150
 334 PRINT TAB 2;B;"#";
 340 PRINT "ITEM NAME?",
 350 INPUT I$(B)
 355 PRINT I$(B)
 360 PRINT TAB 5;"HOW MANY?",
 365 INPUT Q(B)
 368 PRINT Q(B)
 370 LET X=B
 375 IF X=N THEN GOTO 390
 380 NEXT B
 390 CLS
 400 PRINT "IF ""5/???"" APPEARS
     IN THE LOWER LEFT CORNER TY
     PE""C"" TO CONTINUE LISTING."
      [Note: "" is on shift Q]
 402 PRINT
 410 PRINT "ITEM NO.";TAB 10;"IT
     EM NAME"; TAB 24;"QUANT."
 420 FOR B=1 TO 150
 430 PRINT TAB 3;B; TAB 11;I$(B)
     ; TAB 26;Q(B)
 440 LET X=B
 455 IF X=N THEN GOTO 470
 460 NEXT B
 470 PRINT
 471 PRINT "END OF FILE."
 472 PRINT
 475 PRINT "TYPE ""1"" TO RE-DO
     THE INVENTORY LIST, ""2"" TO
     RETURN TO START OF PROGRAM."
 480 INPUT P
 482 CLS
 485 IF P=1 THEN GOTO 300
 490 IF P<>1 THEN GOTO 6
 600 PRINT TAB 7;"INVENTORY LIST
     ING"
*601 PRINT "********************
     ************"
 602 PRINT
 603 PRINT "IF ""5/??"" APPEARS
     IN THE LOWER LEFT CORNER
     TYPE""C""TO CONTINUE##
     LISTING."
 605 PRINT
 610 PRINT "ITEM NO."; TAB 10;"I
     TEM NAME"; TAB 24:"QUANT."
 615 PRINT
 620 FO B=1 TO 150
 622 LET X=B
 630 PRINT TAB 3;B; TAB 11;I$(B)
     ; TAB 26;Q(B)
 635 IF X<>N THEN GOTO 645
 638 PRINT
 640 PRINT "END OF FILE."
 641 GOTO 650
 645 NEXT B
 650 PRINT
 660 PRINT "TYPE""1""TO EXAMINE
     INVENTORY, ""2""TO RETURN TO
     START OF PROGRAM."
 670 INPUT D
 672 CLS
 680 IF D=1 THEN GOTO 600
 690 IF D<>1 THEN GOTO 6
 900 PRINT TAB 6;"ADD ITEM TO IN
     VENTORY"
*901 PRINT "********************
     ************"
 902 PRINT
 906 PRINT "WHEN YOU WISH TO END
     NEW ENTRIESTYPE ""RETURN."""
 910 PRINT
 920 PRINT "IF ""5/???"" APPEARS
     IN THE LOWER LEFT CORNER TYPE
     ""C"" TO CONTINUE##LISTING."
 925 PRINT
 930 PRINT "ITEM NO."; TAB 10;"I
     TEM NAME"; TAB 24;"QUANT."
 935 PRINT
 940 FOR B=1 TO 150
 950 PRINT TAB 3;B; TAB 11;I$(B)
     ; TAB 26:Q(B)
 960 LET X=B
 965 LET G=N
 970 IF B=150 THEN GOTO 1000
 980 IF X=N THEN GOTO 1040
 990 NEXT B
1000 PRINT "SORRY,FILE IS FULL."
1010 PRINT "TYPE ""C"" TO RETURN
     TO START OF PROGRAM."
1020 STOP
1030 GOTO 6
1040 LET B=B+1
1045 LET N=B
1048 IF G=B THEN GOTO 1000
1050 PRINT
1060 PRINT "ITEM NO.";B
1070 PRINT TAB 3;"ITEM NAME? ";
1080 INPUT C$
1081 PRINT C$
1082 LET I$(B)=C$
1084 IF C$="RETURN" THEN GOTO 11
     20
1090 PRINT TAB 3;"HOW MANY? ";
1100 INPUT Q(B)
1105 CLS
1100 GOTO 1040
1120 CLS
1125 LET N=B-1
1130 GOTO 6
1200 PRINT TAB 3;"CHANGE QUANTIT
     Y OF AN ITEM"
*1202 PRINT "********************
        ***************"
1210 PRINT
1212 PRINT "IF ""5/???"" APPEARS
     IN THE LOWER LEFT CORNER TYPE
     ""C"" TO CONTINUE LISTING."
1214 PRINT
1220 PRINT "SELECT ITEM TO BE CH
     ANGED BY ""ITEM NO."""
1222 PRINT
1230 PRINT "ITEM NO."; TAB 10;"I
     TEM NAME"; TAB 24;"QUANT."
1235 PRINT
1240 FOR B=1 TO 150
1260 LET X=B
1270 PRINT TAB 3;B; TAB 11;I$(B)
     ; TAB 26;Q(B)
1280 IF X<>N THEN GOTO 1300
1285 PRINT
1290 PRINT "END OF FILE."
1292 GOTO 1310
1300 NEXT B
1310 PRINT
1320 PRINT "SELECT ITEM TO BE CH
    ANGED BY ""ITEM NO."""
1330 INPUT B
1335 CLS
*1340 PRINT "ITEM NO.#";B;"#IS#";
     I$(B);"#WHICH CURRENTLY
     CONTAINS #";Q(B);"###UNITS."
*1342 PRINT
1350 PRINT "INPUT QUANTITY CHANG
     E."
1351 PRINT
*1352 PRINT "USE A ""MINUS""SIGN
     TO REDUCE THE QUANTITY."
1360 INPUT K
1362 CLS
1370 LET Q(B)=Q(B)+K
*1380 PRINT "ITEM NO.#";B;",#";I$
     (B);",";"NOW HAS#";Q(B);"#U
     NITS.
     "
*1382 PRINT
1390 PRINT "TYPE""1""TO CHANGE A
     NOTHER ITEM OR""2""TO RETUR
     N TO START OF PROGRAM OR ""
     3"" TO REVIEW THIS LISTING."
1400 INPUT R
1405 CLS
1410 IF R=1 THEN GOTO 1320
1420 IF R=2 THEN GOTO 6
1425 IF R=3 THEN GOTO 1200
1500 PRINT TAB 2;"DELETE AN ITEM
     FROM INVENTORY"
*1501 PRINT "********************
     ***********
1510 PRINT
1520 PRINT "EACH ITEM WILL APPEA
     R ONE AT A TIME.""
1530 PRINT
1540 PRINT TAB 2;"1) IF YOU DO NO
     T WANT TO DELETE THE ITEM
     TYPE""S."""
1545 PRINT
1550 PRINT TAB 2;"2)IF YOU WANT
     TO DELETE THE ITEM TYPE ""
     D."""
1555 PRINT
1560 PRINT TAB 2;"3)IF YOU WISH
```

```
      TO TERMINATE ""DELETE"" TY
      PE ""T."""
 1562 PRINT
*1565 PRINT "**********"
*1567 PRINT
*1570 PRINT "ITEM NAME"
*1575 PRINT
 1580 LET X=1
 1590 FOR B=X TO 150
 1600 IF I$(B)="" THEN GOTO 1710
 1610 LET Y=B
 1615 LET X=B
 1618 LET Z=N
 1620 PRINT TAB 3;I$(B);"#";
 1630 INPUT Z$
 1635 PRINT Z$
 1640 IF Z$="S" THEN GOTO 1710
 1650 IF Z$="D" THEN GOTO 1800
 1660 IF Z$="T" THEN GOTO 1675
 1665 CLS
 1670 GOTO 1520
 1675 CLS
 1680 GOTO 6
 1710 IF B=N THEN GOTO 1730
 1720 NEXT B
 1725 PRINT
 1730 PRINT
 1735 PRINT "END OF FILE."
 1740 PRINT
 1750 PRINT "TYPE ""1"" IF YOU WI
      SH TO DELETE OTHER ITEMS,TY
      PE ""2 "" TO RETURN TO
      START OF PROGRAM."
 1760 INPUT W
 1765 CLS
 1770 IF W=1 THEN GOTO 1500
 1780 IF<>1 THEN GOTO 6
 1800 LET I$(B)=""
 1810 LET Q(B)=0
 1815 IF N=B THEN GOTO 1920
 1820 FOR B=Y TO 150
 1830 LET I$(B)=I(B+1)
 1835 LET Q(B)=Q(B+1)
 1840 IF Z=B THEN GOTO 1900
 1850 NEXT B
 1900 LET N=N-1
 1908 LET B=X
 1909 IF N=B THEN GOTO 1730
 1910 GOTO 1590
 1920 LET N=N-1
 1930 GOTO 1730
 2000 PRINT "YOU HAVE INDICATED Y
      OU WISH TO##EXIT THIS PROGR
      AM."
*2005 PRINT
 2010 PRINT "IF YOU HAVE MADE ANY
      CHANGES"
*2012 PRINT
*2013 PRINT "*********************
      *********"
 2015 PRINT "*DO NOT FORGET TO RE
      -LOAD TAPE*"
*2016 PRINT "*********************
      ************"
*2017 PRINT
 2018 PRINT "TO SAVE THIS PROGRAM
      AS CHANGED PREPARE THE TAPE
      RECORDER, BEGIN RECORDING,
      AND TYPE ""C."""
 2020 STOP
 2030 SAVE "PANTRY INVENTORY-10"
 2040 GOTO 6
 3000 PRINT TAB 7;"SEARCH FOR AN
      ITEM"
*3010 PRINT "*********************
      **********"
 3020 PRINT
 3030 PRINT "TYPE THE NAME OF THE
      ITEM YOU##ARE SEARCHING FOR
      IN THE PANTRY."
 3040 INPUT C$
 3042 LET J=LEN C$
 3044 DIM M$(151,J)
 3048 FOR B=1 TO 150
 3052 LET M$(B)=I$(B)
 3068 C$=M$(B) THEN GOTO 3100
 3070 NEXT B
 3075 PRINT
 3080 PRINT "NO SUCH ITEM HAS BEE
      N FOUND IN##THE INVENTORY."
 3090 GOTO 3120
*3100 PRINT
*3102 PRINT
*3104 PRINT
 3110 PRINT "THERE ARE#";Q(B);"#
      UNITS OF"
*3114 PRINT
 3115 PRINT TAB 5:C$
*3119 PRINT "LOCATED IN THE PANTR
      Y"
*3120 PRINT
*3122 PRINT
*3124 PRINT
 3130 PRINT "TYPE ""1"" TO SEARCH
      FOR ANOTHER ITEM, ""2"" TO
      RETURN TO START OF PROGRAM."
 3140 INPUT U
 3145 CLS
 3150 IF U=1 THEN GOTO 3000
 3160 IF U<>1 THEN GOTO 6
```

4K ROM
1K RAM

Part 2

Machine Language Teaches the ZX80 to READ

Edward A. Kennedy, Jr.

In Part 1 (*SYNC* 1:5) we saw what a READ statement is, how it functions, how to run a machine language subprogram with the USR function, and how to get your machine code into memory with Basic loader program.

In this part I will give you the machine language READ subroutine, present a few tips on how to get it running, and describe a couple of the features of the Sinclair system that I made use of when I wrote it. If you typed your loader program in last time and SAVEd it, you should be ready to go.

The READ Subroutine

Listing 1a and 1b show you what your TV will look like when you type the subroutine in. Listing 2 has the same information in the column headed "Machine Language" that listings 1a and 1b have, but Listing 2 has much, much more information for those who want to see how this thing works. It has extensive comments on the overall design, more comments on the individual instructions, and the assembly language. When I program, I figure out what I want to do (the overall design) and how to do it. Then I write that in assembly language, and only as a last step do I convert the assembly language to machine language. All of that is in Listing 2.

A READ Revisited

I started this whole discussion in Part 1 by talking about the need for a method to READ numbers out of special program lines into array elements. In the first Listing I showed what a READ might look like if we had one in this Basic. Now that we have a subroutine that performs the same function, I would like to come the full circle and show how I would adapt that hypothetical listing to function with this subroutine. I think Listing 4 says it all. Try adding these lines to your subroutine and its supporting lines! Do not forget the comma in line 70!

I like this set-up because it looks and functions a lot like the ideal that I am trying to emulate. It does have a few differences, however.

Numbers and Non-Numbers

I said before that the subroutine will not function if the word "DATA" is misspelled or if there are spaces between the letters. READ statements in other Basics are usually sensitive to spelling but not to spacing.

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

Listing 1a. READ subroutine, machine language, first par. When you press NEWLINE after typing in the "25" in the bottom line, your screen will go blank for a moment. Then you will see the first three lines shown in Listing 1b, and the 25 will be the first number in the fourth line.

```
11  08  3F  14  1A  F5  13  1A
F5  D1  2F  F5  E1  F1  06  08
0F  CB  1B  10  FB  2F  06  08
1F  CB  1D  10  FB  23  13  13
D5  D5  1A  F5  13  1A  F5  D1
F1  06  08  1F  CB  1B  10  FB
C1  03  03  0A  0F  38  29  13
E5  19  E1  30  09  C1  01  00
3F  04  3E  01  02  C9  1A  FE
29  20  EC  13  1A  FE  26  20
E6  13  1A  FE  39  20  E0  13
1A  FE  26  20  DA  3E  01  02
21  00  00  13  1A  FE  D8  28
"25L"
```

Listing 1b. A continuation of Listing 1a. Note that the first three lines across the page ar: the last three lines from Listing 1a. Continue typing with the second number in the fourth line of this listing.

```
E6  13  1A  FE  39  20  E0  13
1A  FE  26  20  DA  3E  01  02
21  00  00  13  1A  FE  D8  28
25  06  3B  CB  00  B8  28  18
D6  1C  38  EF  FE  0A  30  EB
29  E5  29  29  C1  09  06  08
1F  CB  19  10  FB  09  18  DB
C1  C5  03  03  AF  02  E3  EB
E3  C1  C5  06  08  CB  19  1F
10  FB  12  F1  13  12  C9
```

Listing 3. Seven lines that can be added to the loader program (Listing 4 in Part 1) to check out the READ subroutine after it has been stored at line 1.

```
  10 REM DATA 1,2,3,4,5,6,7,8,9,
10,11,12
  15 FOR J=1 TO 23
  20 LET L=USR(16427)
  25 PRINT L,U(0),U(1)
  30 NEXT J
  35 STOP

1200 REM DATA 101,102,103,,
```

Listing 4. Using a subroutine (located at 16,427 in RAM) to simulate a READ on the Sinclair with a 4K Basic. Notice that line 50 is a REMark. Also, line 30 does not "READ A(J)" but LETs A(J) equal the value returned by the machine routine.

```
  10 DIM A(50)
  20 FOR J=0 TO 50
  30 LET A(J)=USR(16427)
  40 NEXT J
  50 REM 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
  60 FOR Z=0 TO 50
  70 PRINT A(Z),
  80 NEXT Z
```

This one is like the others in that it ignores spaces between numbers.

```
••50•REM•DATA•17••4••,3••9,3•17,
2,•••••5••5,••7••8,•(etc.)
```

This works fine. In fact, it does not even care if the spaces are turned into letters. Even this:

```
••50•REM•DATA•17WH4AT,3A•9,
317,
2BUNCH5OF5,JU7NK8,•(etc.)
```

does not bother it. You will not find another READ statement anywhere that will READ that line.

Negative Numbers

Just for a test, you might want to add a few minus signs in front of your numbers.

```
50 REM DATA -174,-39,-317,255,
78, (etc.)
```

Shucks! When this is RUN, the subroutine ignores the minus signs just like it ignores the other junk we gave it, and we still get positive numbers. But if we try subtracting each of these numbers from 65,536:

```
65,536 - 174 = 65,362
65,536 -  39 = 65,497
65,536 - 317 = 65,219
65,536 - 255 = 65,281
```

and place the results in our line of DATA where the negative numbers were:

```
50 REM DATA 65362,65497,65219,
65218,78, (etc.)
```

Eureka! It works! But why does it work?

The answer to that lies in the way a computer handles positive and negative numbers. It uses a system called "two's complement representation." If you want to know more, I suggest that you pick up a good book on Z-80 microprocessor programming and look up two's complement. The first of the two books by Nichols, Nichols and Rony (see References in Part 1) has an excellent six-page discussion of two's complement.

System Features That the READ Subroutine Uses

In this project, I made use of two system features, which I will describe.

Finding a Variable

You cannot use the READ subroutine without including the following line in your Basic program:

```
2 DIM U(1)
```

Getting It to Work

1. LOAD the loader program from Part 1.

a. The REMark line has been numbered line 1 to make sure it stays in the same location in RAM.

b. There need to be as many minus signs in line 1 as there are bytes in the program plus maybe one or two extra for safety. This program is 159 bytes long.

c. SAVE the loader for your own experiments in programming.

2. Now, RUN it. Your screen will go blank except for the cursor in quotation marks in the upper left corner.

a. It is looking for a hexadecimal number (0 thru 9 or A thru F) or for the command to stop which is an "S".

b. Always give it two numbers! If you want to enter "8" type "08" and not "8"! Press NEWLINE when you are satisfied with your entry.

c. After you enter the 105th byte, the program will take a moment to erase the top ten lines, and will reprint bytes numbered 81 thru 105. After the 159th byte, the program will automatically stop.

d. To correct an error (ten bytes back for example):

1) Stop the program by typing "S" and NEWLINE;

2) Change the address by typing LET I=I-10 and NEWLINE;

3) Press CONTinue and NEWLINE;

4) When the error has been corrected, you can get back to where you were by using a similar procedure to add nine to I.

3. Now enter the subroutine shown in Listing 1a, and in the bottom seven lines of Listing 1b. The top three lines shown in Listing 1b repeat the bottom three in Listing 1a. This will be helpful if you write your own machine programs.

a. Check your code carefully against the listings. The Sinclair system will not check for errors in machine code like it does in Basic. One error could cause you to bomb and lose the whole program.

b. SAVE a copy right now. If it does bomb because of an error that you misssed, you will not have to start all over.

4. Now it is time to make sure it works.

a. The lines in Listing 3 can be added to test it.

1) The keyword REM in lines 10 and 1200 allows us to type the word DATA, which the subroutine needs.

2) DATA lines have no comma at the end. I put two at the end of line 1200 to show how that messes things up.

b. When you RUN it, check the display as follows:

1) The numbers in the left column are the DATA that was just READ out of lines 10 and 1200. Note the extra zeros at the bottom. These are caused by the commas at the end of line 1200.

2) Each number in the middle column is the address of the comma or EOL (End-Of-Line) marker following the piece of DATA to the left of it in the same row. If your addresses are not the same as mine, do not worry about it.

3) The numbers in the right column are the flags. A one says it came back because it found a comma and there are more numbers in the line. A zero says the subroutine found an EOL and must look for another line of DATA if it is called again.

4) The "2:20" in the lower corner of the screen says that the subroutine ran out of DATA. We asked it to do twenty-three READ operations, but only gave it eighteen numbers to READ.

c. Did your display do all that? If it did not function properly, the problem could be with the Basic or with the machine code.

1) In the Basic the word "DATA" must be spelled correctly with no spaces between the letters. Also, commas between the numbers must be commas and not periods.

2) If that is not the problem problem, check the machine code again.

5. If it works right, you can take out all the lines numbered ten and higher.

a. What you will be left with is the bar subroutine in line 1, the next two lines which it needs to function properly (line 2 and line 3), and line 5, which is not essential but will help to remind you where to find the start of subroutine.

b. If you SAVE a copy or two of this, you will have it ready when you want to convert a program that needs a READ.

Listing 2. The READ subroutine—assembly and machine languages with addresses and comments. (Only relative jumps are used, so addresses are re-locatable).

Listing of a subroutine that simulates a READ statement with the Sinclair 4K Basic. Prepared June 7, 1981, from a hand-assembled source by Edward A. Kennedy.

```
 Location      Machine     Labels   Assembly      Comments
Decml / Hex    Language             Language
```

Point the DE register-pair to 4008 hex (16392 decimal). The 16-bit number stored there at the next location is the vector pointing to the beginning of the user's variable file. Copy it into the DE pair and put its twos-complement into HL.

```
16427 / 402B   11 08 3F             LD DE,3F08H   ;Put 4008H
16430 / 402E   14                   INC D         ;    into DE.
16431 / 402F   1A                   LD A,(DE)     ;Get lo byte of vector
16432 / 4030   F5                   PUSH AF       ;    and save it.
16433 / 4031   13                   INC DE
16434 / 4032   1A                   LD A,(DE)     ;Get hi byte
16435 / 4033   F5                   PUSH AF       ;    and transfer it
16436 / 4034   D1                   POP DE        ;    to D.
16437 / 4035   2F                   CPL           ;And its ones complement
16438 / 4036   F5                   PUSH AF       ;    goes
16439 / 4037   E1                   POP HL        ;    to H.
16440 / 4038   F1                   POP AF        ;Retrieve the lo byte
16441 / 4039   06 08                LD B,08H      ;    rotate it circular
16443 / 403B   0F          LOOP     RRCA          ;    back into itself
16444 / 403C   CB 1B                RR E          ;    and into E
16446 / 403E   10 FB                DJNZ LOOP     ;(FBH = -5 decimal)
16448 / 4040   2F                   CPL           ;And its ones-complement
16449 / 4041   06 08                LD B,08H
16451 / 4043   1F          LOOPO    RRA           ;    rotate that
16452 / 4044   CB 1D                RR L          ;    into L
16454 / 4046   10 FB                DJNZ LOOPO    ;(FBH = -5 decimal)
16546 / 4048   23                   INC HL        ;Add one to get twos-complement
16457 / 4049   13                   INC DE        ;Move DE past the header
16458 / 404A   13                   INC DE        ;    of the first variable (the U array)
16459 / 404B   D5                   PUSH DE       ;    and save two copies
16460 / 404C   D5                   PUSH DE       ;    for later use
```

This should be an array element, and the 16-bit value stored here should point to the next location where we want to look for DATA. We copy it into the DE pair.

```
16461 / 404D   1A                   LD A,(DE)     ;Load lo byte
16462 / 404E   F5                   PUSH AF       ;    and shove it
16463 / 404F   13                   INC DE
16464 / 4050   1A                   LD A,(DE)     ;Load hi byte
16465 / 4051   F5                   PUSH AF       ;    and move it
16466 / 4052   D1                   POP DE        ;    to D
16467 / 4053   F1                   POP AF        ;Retrieve lo byte
16468 / 4054   06 08                LD B,08H
16470 / 4056   1F          LOOP1    RRA           ;    and rotate it
16471 / 4057   CB 1B                RR E          ;    into E
16473 / 4059   10 FB                DJNZ LOOP1    ;(FBH = -5 decimal)
```

Retrieve the array pointer from the stack, add two to it so that it points to the next element (the flag), copy that into A, and test it. A one indicates we are in a line of DATA.

```
16475 / 405B   C1                   POP BC        ;Pop pointer
16476 / 405C   03                   INC BC        ;Point it to second
16477 / 405D   03                   INC BC        ;    array element
16478 / 405E   0A                   LD A,(BC)     ;Load flag byte
16479 / 405F   0F                   RRCA          ;Rotate flag into carry—if one, jump
16480 / 4060   38 29                JR C,JUMP3    ;(29H = +41 dicimal)   to get DATA
```

If flag was zero, search for sequence of letters D-A-T-A. Also test to see that the search has not gone beyond the end of the program. If past end, leave an error message and return. If DATA found, change flag to one.

```
16482 / 4062   13          JUMP1    INC DE        ;Move to next location
16483 / 4063   E5                   PUSH HL       ;Subtract address of end of program
16484 / 4064   19                   ADD HL,DE     ;    by adding its twos-complement
16485 / 4065   E1                   POP HL        ;    —preserve HL in process
16486 / 4066   30 09                JR NC,JUMP2   ;(09H = +9D)  Still OK if no carry
16488 / 4068   C1                   POP BC        ;Else straighten the stack,
16489 / 4069   01 00 3F             LD BC,3F00H
16492 / 406C   04                   INC B         ;    leave error message
16493 / 406D   3E 01                LD A,01H      ;    (a one)
16495 / 406F   02                   LD (BC),A     ;    at 400CH,
16496 / 4070   C9                   RET           ;    and go back
16497 / 4071   1A          JUMP2    LD A,(DE)     ;Search comes here if still in program
16498 / 4072   FE 29                CP 29H        ;Look for "D"
```

```
 Location      Machine     Labels   Assembly      Comments
Decml / Hex    Language             Language

16500 / 4074   20 EC                JR NZ,JUMP1   ;(ECH = -20D)   and go back if not
16502 / 4076   13                   INC DE
16503 / 4077   1A                   LD A,(DE)     ;Else look next for "A"
16504 / 4078   FE 26                CP 26H
16506 / 407A   20 E6                JR NZ,JUMP1   ;(E6H = -26D)   and go back if not
16508 / 407C   13                   INC DE
16509 / 407D   1A                   LD A,(DE)     ;Else look next for "T"
16510 / 407E   FE 39                CP 39H
16512 / 4080   20 E0                JR NZ,JUMP1   ;(E0H = -32D)   and go back if not
16514 / 4082   13                   INC DE
16515 / 4083   1A                   LD A,(DE)     ;Else next look for "A"
16516 / 4084   FE 26                CP 26H
16518 / 4086   20 DA                JR NZ,JUMP1   ;(DAH = -38D)   and go back if not
16520 / 4088   3E 01                LD A,01H      ;Found?  BC still points to flag in
16522 / 408A   02                   LD (BC),A     ;    user array—change flag
```

Come here (from either of two places) after deciding that we have a line of DATA. Put zero in sum—the HL register pair.

```
16523 / 408B   21 00 00    JUMP3    LD HL,0000H   ;zero sum
```

Get next character. Jump to return routine when comma or End-Of-Line is found. If it is a number, multiply sum by ten, add number to it, and come back for next character.

```
16526 / 408E   13          JUMP4    INC DE
16527 / 408F   1A                   LD A,(DE)     ;Load next character
16528 / 4090   FE D8                CP D8H        ;Check for comma
16530 / 4092   28 25                JR Z,JUMP6    ;(25H = +37 decimal)
16532 / 4094   06 3B                LD B,3BH      ;Else put End-Of-Line
16534 / 4096   CB 00                RLC B         ;    into B
16536 / 4098   B8                   CP B          ;    and check for it
16537 / 4099   28 18                JR Z,JUMP5    ;(18H = +24 decimal)
16539 / 409B   D6 1C                SUB 1CH       ;Convert it to a number (SUB 28D)
16541 / 409D   38 EF                JR C,JUMP4    ;(EFH = -17D) Less than zero? Go back
16543 / 409F   FE 0A                CP 0AH        ;Is it a letter?  (Greater than nine?)
16545 / 40A1   30 EB                JR NC,JUMP4   ;(EBH = -21 decimal)  Then go back
16547 / 40A3   29                   ADD HL,HL     ;Multiplication by ten
16548 / 40A4   E5                   PUSH HL       ;    requires six commands
16549 / 40A5   29                   ADD HL,HL
16550 / 40A6   29                   ADD HL,HL
16551 / 40A7   C1                   POP BC
16552 / 40A8   09                   ADD HL,BC
16553 / 40A9   06 08                LD B,08H      ;Now rotate
16555 / 40AB   1F          LOOP2    RRA           ;    latest number into C
16556 / 40AC   CB 19                RR C          ;    (B will be zero when loops done)
16558 / 40AE   10 FB                DJNZ LOOP2    ;(FBH = -5 decimal)
16560 / 40B0   09                   ADD HL,BC     ;Add it in then go get next character
16561 / 40B1   18 DB                JR JUMP4      ;(DBH = -37 decimal)
```

If subroutine comes here, DATA is already in HL register pair. The return has two parts. The first part changes the flag in the second array element when an End-Of-Line was found (not a comma).

```
16563 / 40B3   C1          JUMP5    POP BC        ;Top stack entry points at first array
16564 / 40B4   C5                   PUSH BC       ;    element.  Increment it twice
16565 / 40B5   03                   INC BC        ;    to point at second.
16566 / 40B6   03                   INC BC
16567 / 40B7   AF                   XOR A         ;Put zero in A
16568 / 40B8   02                   LD (BC),A     ;    and then in flag.
```

Second part of return puts address of the End-Of-Line or comma into first array element and then returns. The triple swap in the first three lines moves the address of the comma or EOL from DE to the stack and the address of the first array element from the stack to DE while leaving the DATA in HL intact.

```
16569 / 40B9   E3          JUMP6    EX (SP),HL    ;DATA, DATA,
16570 / 40BA   EB                   EX DE,HL      ;    who's got
16571 / 40BB   E3                   EX (SP),HL    ;    the DATA?
16572 / 40BC   C1                   POP BC        ;Copy lo byte of comma or EOL
16573 / 40BD   C5                   PUSH BC       ;    address into C,
16574 / 40BE   06 08                LD B,08H
16576 / 40C0   CB 19       LOOP3    RR C          ;    move it
16578 / 40C2   1F                   RRA           ;    to A,
16579 / 40C3   10 FB                DJNZ LOOP3    ;(FBH = -5D)
16581 / 40C5   12                   LD (DE),A     ;    and store it.
16582 / 40C6   F1                   POP AF        ;Retrieve hi byte
16583 / 40C7   13                   INC DE
16584 / 40C8   12                   LD (DE),A     ;    and store it
16585 / 40C9   C9                   RET           ;Return to the BASIC with DATA
```

This line creates an array with two elements in it, U(0) and U(1). The READ sub uses both these elements. But there must first be an array where it expects to find one. It does not matter what letter the array is identified by (it does not have to be "U"), nor does it matter if it has more than two elements (the extra ones will not be used). But there must be an array starting at the very first locations of the space used to store variables.

Once a variable is created, it will reside in the user-variables section of RAM until it is erased by a CLEAR or a RUN. The rule, except for string variables, is that the first variable created is in the first locations of that section. (Strings are erased from earlier locations and moved to the end every time they are changed.) The newest variable always goes to the end of the file.

Now the user-variables section begins where the program ends. This means that the whole section is going to get moved around. That's right! When you add a line to your program, it gets longer, and the Sinclair system knows that it must move the variables to make room. Likewise, it moves everything together again, to fill the gap, when you take a line out.

But do not think that it does all this blindly. It does not. In fact, it keeps accurate track of where everything is. It reserves ten locations, in the lower part of RAM, to help it do that. Two of these locations store the low part and the high part *of the address of the beginning of the first byte* OF THE FIRST USER VARIABLE.

My READ subroutine finds that address, presumes the first variable is going to be an array, and goes to the place where the first element would be (if it actually were an array). It expects to find an address there pointing to the place in the program where it has to look for more DATA. It also expects a second element to hold a flag telling it whether that address is inside a line of DATA or not. If not, it knows that it will have to initiate a search for a place farther down in the program where the word "DATA" has been typed.

Then, before it returns to execute the rest of the Basic line, it updates those two things that it thinks are array elements.

As I said, it only thinks that the array is in the right spot. It does not do any checking. The programmer who wants to use this subroutine is going to have to make sure that a two-element array gets placed at the very beginning of the variable file. This is easy to do. Just make sure that the first variable created is the needed array. Then you will know it is in the beginning. Even though the system will move the file around as you make your program larger or smaller, there will always be a pointer (at locations 16392 and 16393) to the beginning of the file and thus to the special array.

DIMension statements create array type variables. LET and FOR statements also create variables some of the time. LET opens space for simple variables when they do not already exist. FOR does the same for the type of variable that controls a FOR-NEXT loop—it creates one when it did not exist before.

Thus, if, when the computer is RUN, it finds the DIMension statement before it finds any FORs or LETs, I know things will function correctly. This means that I make line 2 a DIMension. I could have added some sort of checking routine, but that would have lengthened the subroutine. I would rather put up with the quirk of having to place the thing properly than to make it any bigger than it is.

Error Messages

One added feature is a subroutine to stop the Basic program with an error message when it runs out of DATA.

The first location in RAM is used as a mail box for error messages. Each time it executes a line, the system checks its "mail" to see if any mistakes were made. If it finds the number 255 there, it knows everything is OK and keeps going. If it finds a number from zero through eight, however, there was an error. So it stops the program, adds one to that number, and displays it, together with the line number where the error was found, in the lower left corner of your TV screen.

The location of the mail box is 4000 hex (or 16384 decimal—see Appendix: System Memory Locations in your manual for more details). If you do not type in enough DATA, the subroutine will signal you when it runs out. It will put a 1 in the mail box and the system will stop the Basic. That means you will usually see a 2 followed by a colon and the line number where you were trying to use the USR function. (I have seen it change to a three when I was printing an array element in the same line. For some reason, the system changed it.)

Getting by Without the "Forbidden Codes"

The remainder of the article presumes a knowledge of machine language programming, preferably on the Z80.

As noted Part 1, there are many numbers that cannot be POKEd into line 1 without messing things up. Those are the numbers from 40 hex through 7F hex. This restriction takes certain instructions away from us outright, and places severe limitations on our use of other ones. The instructions that are totally forbidden are the ones whose machine code number falls in the forbidden group. The ones that it restricts are those that sometimes require data in that range.

1) *Forbidden Instructions (Single Op Code)*

The single byte instructions that cannot be used in this system are transfers from one register to another within the CPU, and the transfers between a register and a memory location when using the HL register pair as the pointer to that location. Fortunately, the accumulator (the A register) can transfer to or from memory using either the BC pair or the DE pair as pointers. Those op codes (operation codes) are not forbidden. They are:

02 for LD (BD),A	copy accumulator into memory at address in BC;
0A for LD A,(BC)	copy memory at address in BC into accumulator;
12 for LD (DE),A	copy accumulator into memory at address in DE; and
1A for LD A,(DE)	copy memory at address in DE into accumulator.

They work fine. We can use them to get information between the accumulator and any memory location in the computer. If we could find a way to transfer it between the accumulator and the register that is to be its destination (or soure), then we would have it.

We have two ways of working this. If we want to transfer a value between the accumulator and the high byte of a register-pair, we can put it on the stack and take it off again. For instance, to transfer a value from A to D, we can put the accumulator and flags onto the stack and take it off again into DE.

Example 1.

```
••••••F5••••••••••••••PUSH AF
      D1              POP DE
```

This copies the accumulator into the D register and the flags into the E register. Going the other way:

Example 2.

```
••••••D5••••••••••••••PUSH DE
      F1              POP AF
```

copies the D register into the accumulator, but you should remember that it also changes the flags. Normal register transfers do not change the flags.

For other moves we can use the rotate instructions. One rotate will move a single bit from one register into the carry flag, and another rotate will get that bit from the flag into the other register. Then we repeat that operation seven more times on the next seven bits, and we have moved the contents of one register into the other. Each of the following three examples puts a copy of the accumulator into the E register:

Example 3.

```
        ••••••5F••••••••••••••LD•E,A
```

Example 4.

```
        ••••••0F••••••••••••••RRCA
              CB 1B           RR E
              OF              RRCA
              CB 1B           RR E
              OF              RRCA
              CB 1B           RR E
              OF              RRCA
              CB 1B           RR E
              OF              RRCA
              CB 1B           RR E
              OF              RRCA
              CB 1B           RR E
              OF              RRCA
              CB 1B           RR E
              OF              RRCA
              CB 1B           RR E
```

Example 5.

```
              06 08           LD B,08H
              OF      LOOP    RRCA
              CB 1B           RR E
              10FB            DJNZ LOOP
```

The first example is the most straight forward. It takes up only one byte of memory, but its op code could mess us up.

Example 4, the second one, gets around that problem, but it requires twenty-four bytes of storage.

The final example requires only seven bytes and uses no objectionable op codes. It uses the DJNZ (Decrement and Jump if Not Zero) instruction to repeat the instructions that move the data, one bit at a time, from the A register, through the carry, and into the E register. In order to make the DNJZ instruction work, we must load the B register with the number eight. DNJZ decrements B, checks to see if it is zero, and jumps if it is not. So the loop will be done eight times. Note that this method changes both the B register and the carry flag, so if either holds important information, you will have to save it and restore it afterwards.

2) *Forbidden Instructions (Multiple Op Code)*

The same restrictions apply to the IX and IY registers as to the HL pair. You cannot use them to load the other CPU registers from memory or vice versa without using one of these numbers. In fact, the second op code in an IX or IY instruction is always the same as the first op code in the similar HL instruction.

The BIT test instructions are all lost because of these restrictions, and so are the INput and OUTput instructions that are distinctive to the Z80. The original 8080 style IN A,(N) and OUT (N),A can still be used.

The only way that I see to replace the BIT test is to rotate the bit to be tested into the carry flag and test the flag. This changes the register or the memory location that you rotate, so you may need to copy the byte into the accumulator and rotate it instead.

To NEG (negate the accumulator two's complement style) you must CPL (one's complement) and then add one:

Example 6.

```
        ••••••ED•44•••••••••••NEG
```

is replaced by:

Example 7.

```
              2F              CPL
              C6 01           ADD 01H
```

These two examples leave the same numbers in the accumulator, although the flags will usually be different.

Sixteen-bit subtraction is lost, so when I needed to do one to find out if the search for a line of DATA had taken us beyond the end of the program, I did it the same way that early programmers had to do their subtracts. I took the two's complement of the subtrahend (the number with the minus in front of it) and I *added* that to the minuend (the other one).

There are a few other instructions that cannot be used, but I had no need for them, so I did not investigate them. These include rotates that move a whole digit (four bits) at a time and transfers to and from the interrupt (I) register or the refresh (R) register.

3) *Forbidden Data*

Early in the subroutine, I wanted to load 4008 hex into the DE pair. I could not do that because it would have put a "40" into my REMark line. I had to put 3F08 hex into DE and increment D.

Later, I wanted to test to see if the subroutine had just loaded an EOL marker. That would have meant that it had come to the end of the line of DATA in the Basic program. But the Sinclair uses 76 hex (or 118 decimal) as EOL signals and I could not just write:

Example 8.

```
        ••••••FE•76•••••••••••CP 76 H
```

because that would have scrambled things. So first I figured out that half of 76 hex is 3B hex. I loaded 3B hex into B, I doubled it by rotating it to the left, and I compared the contents of the accumulator to it.

Example 9.

```
              06 3B           LD B,3BH
              CB 00           RLC B
              B8              CP B
```

I cannot call a subroutine that I have stored somewhere in RAM, and I cannot jump absolute to any location in RAM without writing one of these numbers. RAM begins at 4000 hex and ends at 43FF hex if you have 1K. So the high-order byte of any address in RAM will be from 40 hex through 43 hex. When I tell it to jump absolute or to call, I must give it an address and that high byte is forbidden. That will not change if I get 16K because it will go from 4000 hex through 7FFF hex. If I find some useful subroutines in the system ROM (Read Only Memory), I can call them as long as the low-order byte is not forbidden. But within any program that I intend to store in a REMark line, I am otherwise limited to the relative style of jump.

The relative jumps are the ones in which you do not tell it what *exact address* to go to, but how many spaces to forward or backward. The machine takes that number and the present value of the program counter, then generates a *computed address*, and finally goes to the computed address. Those of us who learned to write machine programs on the 6502 or the 6800 had to learn this relative type of addressing. We could not call *if carry* or jump *if zero* or return *if parity odd.* We had a single, unconditional jump, and a single, unconditional jump to subroutine, and a single, unconditional return. If we did not want the instruction to execute under certain conditions, we would have to put a branching test ahead of it. Then we would avoid the instruction entirely when the conditions were not fulfilled. We were forced to learn how to compute a relative jump. Those of you who have never learned that, can consider this your big headache or your big opportunity!

4) *Computing a Relative Jump.*

None of the books listed at the end of Part 1 really tells you how to figure a relative jump. So, here goes!

A relative jump can go forward 127 locations or backward 128 locations *from the first byte of the first instruction after the jump instruction.*

When I compute the displacement for a relative jump, I first leave the space of one byte after the jump instruction. This is where the displacement will go. I put my finger on the byte after that. This is the first instruction after the jump, and I call it byte number zero. A common mistake is to use the jump itself as byte zero and that will get you in trouble.

The second step is to count the number of memory locations *from there* that I want it to jump when it does jump. (If that is more than 127 forward or 128 backward, I quit and look for another method.)

Third, if the jump is forward, I convert it to hex and that is my relative displacement.

Fourth, if the jump is backward, I subtract the number I got in the second step from 256. My displacement will be the hex version of that difference.

Fifth, I double-check my work by counting the bytes again, usually in hexadecimal this time. I may also check it by putting the segment of code into the computer and running it.

Let's look at a couple of displacements calculated for the READ subroutine presented earlier in this article. The first is a backward jump (Example 10) and the second is a forward jump (Example 11).

Example 10: Backward Jump

```
•••••06 08•••••••••••LD B,08H
     0F              RRCA
     CB 1B           RR E
     10 ??           DJNZ LOOP
     2F              CPL
```

In example 10, when the B register finally decrements to zero, there will be no more jumps and the next instruction will be CPL. So I put my finger on the "2F" and say that it is byte zero.

Second, I count back. The space I have left for the displacement byte (the "??") is number one, "10" is number two, "1B" is three, "CB" is four, and "OF" is number five. That is the byte which I want it to do next when it does jump, so I am done with step two.

It is a backward jump so step three does not apply.

Step four is to subtract the number I got in step two from 256. That leaves me 251. Then I convert it to hex (see below). That gives me FB. So I take the question marks out of the displacement byte and put in "FB".

Fifth, I put my finger back on the "2F" which I called byte zero before. This time I call it byte number FB. Then the displacement byte that I just put in will be number FC, the "10" is FD, the "1B" is FE, the "CB" is FF, and the destination (the "OF") is byte number 100. That is what I wanted to see, because I know that the Z80 CPU is going to do something similar and come to the conclusion that it must execute the "OF" next when it jumps.

Example 11: Forward Jump

```
•••••30 ??•••••••••••JR NC,JUMP2
     C1              POP BC
     01 00 3F        LD BC,3F00H
     04              INC B
     3E 01           LD A,01H
     02              LD (BC),A
     C9              RET
     1A     JUMP2    LD A,(DE)
```

In this case, the ZX80 is to go to the line marked with the label "JUMP2" whenever it does jump. How far is that?

Well, for the first step, I know that the "C1" following the "??" is byte zero. I also know that the "1A" in the bottom line is the destination.

So for the second step, we count. "01" is byte one, "00" is two, "3F" is three, "04" is four, "3E" is five, "01" is six, "02" is seven, "C9" is eight, and "1A" is nine. Nine is the number.

Step three, since nine is the same in hex as it is in decimal (no conversion is needed until the number is ten or larger), I just plug "09" in where I have the "??".

Step four does not apply, so I check my work and I am done.

5) *Converting Decimal to Hexidecimal*

I showed you, in Part 1, that it is easy to convert from hex to decimal. This time I need to show you how to go the other way. But in order to tie it all together, I will review briefly.

First, remember that there are sixteen numbers in the hex system:

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

Let's take the number FB in hex and find out what it is in decimal. We see that an F is equal to our number fifteen. Since every increment in that column is equal to sixteen in the decimal system, an F there equals fifteen times sixteen, or 240. Now the B in the right column is equal to our eleven, and each increment in that column is worth only one, so that is eleven times one, or eleven. Add them together and we have 251.

(Fx16)+(Bx1)=(15x16)+(11x1)=251

End of review. Now let's go the other way—from decimal to hex. If I have a number small enough to fit into a two digit hex number after conversion, then I can make the switch with a single division by sixteen. (255 is the largest decimal number that will work.) In example 10, I ended up needing to convert 251. So I divided:

251/16 = quotient of 15 and remainder of 11

Then, if the quotient or the remainder is ten or more, I substitute the number between A and F that it corresponds to. 15 is the Fth number in the hex sequence, and 11 is the Bth one. The remainder goes in the ones column and the quotient goes in the next (the "sixteens" column. Therefore, the hex equivalent of 251 will be FB.

If you want to practice doing a few jump displacements, you will notice that in the comments column in Listing 2 I give every displacement I computed both in the positive or negative numbers from the steps two, and in the final hexidecimal number.

This explanation goes deep enough to help you convert your jump displacements. The programming texts that I already recommended explain the conversion process in more detail. They will help you convert larger numbers, like addresses, if you need to.

Conclusion

This article has covered a lot of ground. I have given you an implementation of the standard Basic statement "READ" for your Sinclair or MicroAce. Part of my reason for writing this article was to give you an idea of what it is like to program in machine language on these machines. I mainly concentrated on the special techniques needed for that task, and I left alone the information that is already well covered in the programming texts.

I hope you have enjoyed it. Good READing! Good programming.

4K ROM

1K RAM

Hampson's Plane

M. Hampson

Rubik's Cube has presented puzzle solvers a most challenging puzzle, and it raised for me the further question: Was it possible to simulate on the ZX80 at least some of the features of the cube? In the first place we have only two dimensions to work with. We have the limitation of a 4K ROM, a 1K RAM, and a TV screen for display.

The program below is my attempt, after many hours of work, to translate the three dimensional Rubik's Cube to the two dimensional TV screen via the ZX80. The result, at the editor's suggestion, is "Hampson's Plane."

After entering the program, which is rather short, you must take care of a bug in this way: Press RUN and NEWLINE. The prompt will call for a skill level entry; enter 1 and NEWLINE. Wait for the screen display to appear. Press NEWLINE. The prompt will ask for a numerical input; however, enter END and NEWLINE. An error code will appear. Press NEWLINE again to get back to program and you are now ready to start playing. The first thing to do is *save the program*. From now on you must *always* start with GOTO 1. Never use RUN or CLEAR.

The gaming board will appear on the screen. It consists of a board 21 x 15 with video and inverse video crosses. These are identified by the use of the coordinates: letters across the top and numbers down the left side. Type in the coordinates of any cross on the board: enter the letter, NEWLINE, the number, NEWLINE. The video display changes, i.e., from normal to inverse or inverse to normal. That may sound easy, but there is more. Not only does that particular cross change, but so do the surrounding 8 crosses. See Figure 1. The object of the game is to clear the board completely to normal video crosses. While that may sound hard, it is possible. You have your choice of 100 skill levels.

The computer sets out the gaming board and then sets up the problem by choosing some random coordinates according to your skill level. This ensures two things: first, it is *always* possible to complete the puzzle, and, second, it is only as hard as you want it to be. While skill level 100 is very, very hard and can take the best part of an hour to work out, you always know that it is possible because the computer and you got into that mess simply by typing coordinates. But be warned! It takes the computer a long time to set up the board on a high skill level, e.g., allow 1 min. and 45 sec. on level 100.

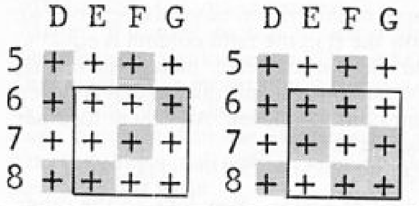
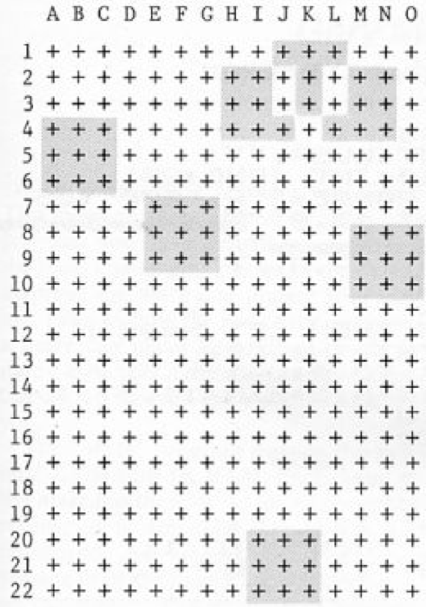


Figure 1.

```
   2 RANDOMISE
   3 PRINT "HAMPSON'S PLANE"
   4 PRINT "ENTER SKILL LEVEL"
   5 INPUT SK
   6 CLS
  10 PRINT "##ABCDEFGHIJKLMNO"
  15 FOR Z=1 TO 22
  20 IF Z<10 THEN PRINT "Ø";
  21 PRINT Z;"+++++++++++++++";
  22 IF Z<22 THEN PRINT
  30 NEXT Z
  40 FOR L=1 TO SK
  50 LET X=RND(13)+3
  60 LET Y=RND(20)+1
  70 GO SUB 1000
  80 NEXT L
 100 INPUT K$
 110 LET X=CODE(K$)-35
 120 INPUT Y
 170 GO SUB 1000
 180 GO TO 100
1010 FOR T=1 TO 9
1020 IF T=1 THEN LET P=0
1021 IF T=2 THEN LET P=1
1022 IF T=3 THEN LET P=-1
1023 IF T=4 THEN LET P=-18
1024 IF T=5 THEN LET P=-17
1025 IF T=6 THEN LET P=-19
1026 IF T=7 THEN LET P=18
1027 IF T=8 THEN LET P=19
1028 IF T=9 THEN LET P=17
1033 LET V=PEEK(16396)+256*PEEK(
16397)
1035 LET N=V+Y*18+X+P
1040 POKE N, PEEK(N)+128
1045 NEXT T
1050 RETURN
```

M. Hampson, 7 Hereford Drive, Clitheroe, Lancs
BB7 1JP, United Kingdom.

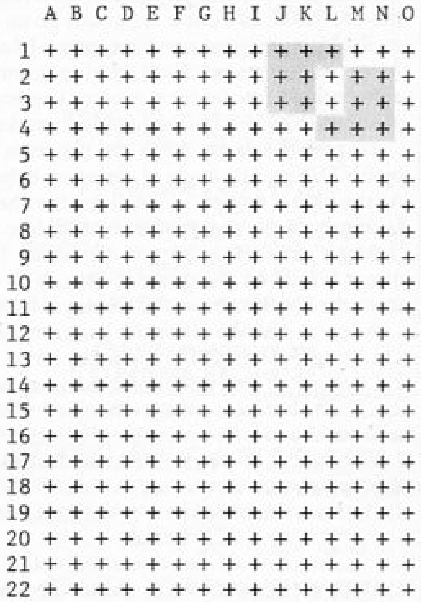


```
    A B C D E F G H I J K L M N O
 1  + + + + + + + + + + + + + + +
 2  + + + + + + + + + + + + + + +
 3  + + + + + + + + + + + + + + +
 4  + + + + + + + + + + + + + + +
 5  + + + + + + + + + + + + + + +
 6  + + + + + + + + + + + + + + +
 7  + + + + + + + + + + + + + + +
 8  + + + + + + + + + + + + + + +
 9  + + + + + + + + + + + + + + +
10  + + + + + + + + + + + + + + +
11  + + + + + + + + + + + + + + +
12  + + + + + + + + + + + + + + +
13  + + + + + + + + + + + + + + +
14  + + + + + + + + + + + + + + +
15  + + + + + + + + + + + + + + +
16  + + + + + + + + + + + + + + +
17  + + + + + + + + + + + + + + +
18  + + + + + + + + + + + + + + +
19  + + + + + + + + + + + + + + +
20  + + + + + + + + + + + + + + +
21  + + + + + + + + + + + + + + +
22  + + + + + + + + + + + + + + +
```

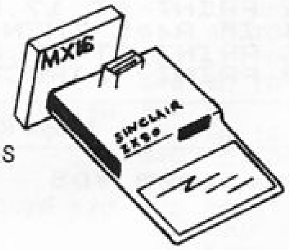
If you have expansion memory, you can do the de-bugging in this way: Set up a flag F thus:
1 LET F=1
Then insert the program alteration which is only required once before the display file has adjusted its position:
1036 IF F THEN LET N=N+6
And finally reset the flag F:
1037 LET F=0
You can now use RUN instead of GOTO 1.

Sample Run
Type GOTO 1
HAMPSON'S PLANE
ENTER SKILL LEVEL
Enter 7 and wait for the display.
The obvious moves are B,5; F,8; N,9 and J,21. But we are left with a muddle at the top. The solution is I,3, K,2, M,3. Now to return to command mode, type NEWLINE, END, NEWLINE. Now try a high skill level. This may keep you going as long as the original cube! So I suggest building up slowly, learning the tricks as you go along. A good plan is to raise your skill level by fives each time you run the program.

The best of British luck!



```
    A B C D E F G H I J K L M N O
 1  + + + + + + + + + + + + + + +
 2  + + + + + + + + + + + + + + +
 3  + + + + + + + + + + + + + + +
 4  + + + + + + + + + + + + + + +
 5  + + + + + + + + + + + + + + +
 6  + + + + + + + + + + + + + + +
 7  + + + + + + + + + + + + + + +
 8  + + + + + + + + + + + + + + +
 9  + + + + + + + + + + + + + + +
10  + + + + + + + + + + + + + + +
11  + + + + + + + + + + + + + + +
12  + + + + + + + + + + + + + + +
13  + + + + + + + + + + + + + + +
14  + + + + + + + + + + + + + + +
15  + + + + + + + + + + + + + + +
16  + + + + + + + + + + + + + + +
17  + + + + + + + + + + + + + + +
18  + + + + + + + + + + + + + + +
19  + + + + + + + + + + + + + + +
20  + + + + + + + + + + + + + + +
21  + + + + + + + + + + + + + + +
22  + + + + + + + + + + + + + + +
```



8K ROM
1K RAM

Artillery with Motion

Chuck Dawson

Ever since seeing the *Artillery* game in the March/April issue of *SYNC*, I have been interested in devising a way for the player to see the projectile on its flight to the target, using the 8K ROM, of course. It would be an easy project with lots of RAM, but it was not that easy when limited to 1K RAM. After rewriting much of the program to save space and eliminating a few features such as keeping track of the number of shells fired, I was finally able to make it work.

To play the game, RUN the program. Make your first shot by ENTERing any angle of elevation between 0 and 90 degrees. At the higher angles, you will see the shell arch way up before *falling back to the ground, much like a mortar shell being* "lobbed" toward its target. Continue firing until you get a direct hit. Then the game is over. You can count your shots from the shell holes in the display. Hit RUN to start over.

A few notes will make the program clearer:

1) Listing 2: l. 12, INPUT is a key word.

2) Listing 2: If you encounter memory problems, omit lines 1, 2, 12, 38, but you must remember what the game calls for without the screen prompts.

3) Listing 2: l. 38, Direct hit must be followed by 5 spaces and the " mark.

4) Both versions: 8, inverse 0, 30 graphic D
32, asterisk
36, inverse asterisk is a hit.

5) Both versions show the projectile; the ZX80 by a PAUSE routine, and the ZX81 by the SLOW mode.

Chuck Dawson, 6520 Victoria, Fort Worth, TX 76118. Program adapted to ZX81 by David Grosjean.

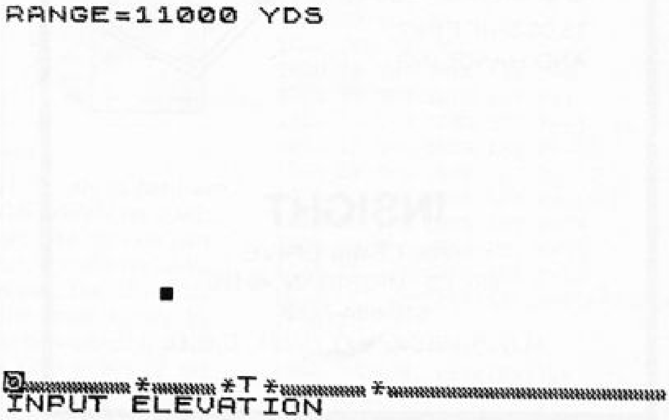
Listing 1. ZX80 8K ROM

```
 1 REM ZX80
 2 REM "ARTILLERY""
 4 LET A=10+INT (RND*20)
 6 PRINT "RANGE=";A;"000 YDS"
 8 PRINT AT 17,0;"▣▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
"
10 PRINT AT 17,A;"T"
14 INPUT N
16 LET B=N
18 IF N>45 THEN LET N=90-N
20 LET L=INT (N*.68)
22 FOR I=1 TO 2*L-1
24 LET J=10+B/3*SIN (1.57*I/L)
25 PLOT I,J
26 PAUSE 20
27 POKE 16437,255
28 UNPLOT I,J
30 NEXT I
32 PRINT AT 17,L;"*"
34 IF A<>L THEN GOTO 14
36 PRINT AT 17,L;"▣"
```

Listing 2. ZX81 SLOW Mode

```
 1 REM ZX81
 2 REM "ARTILLERY""
 4 LET A=10+INT (RND*20)
 6 PRINT "RANGE=";A;"000 YDS"
 8 PRINT AT 17,0;"▣▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
"
10 PRINT AT 17,A;"T"
12 PRINT " INPUT ELEVATION"
14 INPUT N
16 LET B=N
18 IF N>45 THEN LET N=90-N
20 LET L=INT (N*.68)
22 FOR I=1 TO 2*L-1
24 LET J=10+B/3*SIN (1.57*I/L)
25 PLOT I,J
28 UNPLOT I,J
30 NEXT I
32 PRINT AT 17,L;"*"
34 IF A<>L THEN GOTO 14
36 PRINT AT 17,L;"▣"
38 PRINT "DIRECT HIT     "
```

Sample Run



```
RANGE=11000 YDS




                ■


▣▒▒▒▒▒▒*▒▒▒▒*T*▒▒▒▒▒▒*▒▒▒▒▒▒▒▒▒▒▒▒▒
INPUT ELEVATION
```

4K ROM
1K RAM

"You Are in a Maze..."

Gary McGath

"...of twisty little passages, all alike." This message, in *Adventure of Zork*, tells you that you have embarked on one of the most challenging phases of the game: mapping the maze and finding your way out. Even without the rest of the dungeon, a maze can make an exciting puzzle in itself—especially if there is a hungry dragon wandering through the maze looking for you.

The "Maze" program runs on the minimal ZX80 with 1K RAM. Each room of the maze has three doors leading to other rooms, or to freedom. The rooms are numbered, and you know which room you are in by the number that is displayed on the screen. You start off in room 1, just as the dragon is coming in through the exit. If the dragon is in an adjacent room, you can hear it.

You move by entering a 1, 2, or 3 to pick one of the doors, or you can stay where you are by typing a zero. The dragon always moves after you. If you should meet the dragon, you become his dinner. To win, you must find an exit before the dragon finds you.

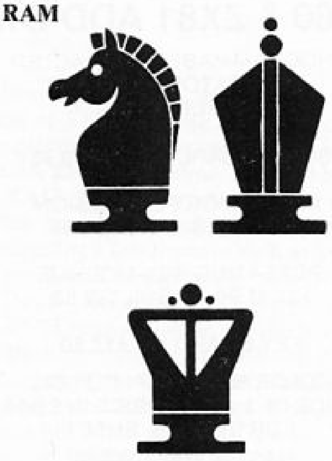
A couple of pointers: (1) Since the dragon moves after you, you may both walk into the same room without your having previously heard the dragon. Thus, you can get eaten without warning. (2) It is not cheating to study the maze generation part of the program for any peculiarities that might help you. For example, it turns out that lower numbered doors tend to lead to lower numbered rooms. (3) Collectors of programming tricks should note the use of the "computed GO TO" in statement 450, in which the next statement executed is determined by a random number. This capability is a feature of ZX80 Basic and does not exist on many other Basics.

May you always elude the dragon in your wanderings.

Gary McGath, 5 Ames Rd., RFD #3, Milford, NH 03055.

```
5    RANDOMISE
10   DIM A(16)
20   DIM B(16)
30   DIM C(16)
40   FOR I = 1 TO 16
50   LET A(I) = 0
60   LET B(I) = 0
70   LET C(I) = 0
80   NEXT I
90   FOR I = 1 TO 16
100  IF A(I)>0 THEN GO TO 140
110  GOSUB 800
130  IF NOT N=I THEN
      LET A(I)=N
140  IF B(I)>0 THEN GO TO 170
150  GOSUB 800
160  IF NOT N=I THEN
      LET B(I)=N
170  IF C(I)>0 THEN GO TO 200
180  GOSUB 800
190  IF NOT N=I THEN LET
     C(I)=N
200  NEXT I
300  LET U=1
315  LET Z=RND(15)+1
316  LET D=Z
320  CLS
330  PRINT U
350  IF NOT(D=A(U) OR D=B(U)
     OR D=C(U))THEN GO TO 380
360  PRINT "YOU HEAR THE
     DRAGON"
380  INPUT X
400  IF X=1 THEN LET U=A(U)
410  IF X=2 THEN LET U=B(U)
420  IF X=3 THEN LET U=C(U)
430  IF U=D THEN GO TO 600
440  IF U=Z THEN GO TO 700
450  GO TO 480+(20*RND(3))
500  LET D=A(D)
510  GO TO 550
520  LET D=B(D)
530  GO TO 550
540  LET D=C(D)
550  IF NOT U=D THEN GO TO 320
600  PRINT "THE DRAGON CAUGHT
     YOU"
610  STOP
700  PRINT "YOU GOT AWAY"
710  STOP
800  LET N=RND(16)
810  IF A(N)=0 THEN GO TO 860
830  IF B(N)=0 THEN GO TO 870
840  IF C(N)=0 THEN GO TO 930
850  GO TO 800
860  LET A(N)=I
870  RETURN
890  LET B(N)=I
900  RETURN
930  LET C(N)=I
940  RETURN
```

4K ROM
1K RAM



The Hidden Chessmen

Roger and Susan Haar

The Hidden Chessmen is a search and find game like *Hurkle* (*SYNC* 1:1, p. 12). A knight, a bishop, and a rook are hidden on a chessboard by the computer. You begin the play by guessing a square on the chessboard. The computer tells you if you have found a chess piece and/or what pieces are attacking the square. From this information you deduce your next guess, and eventually the locations of the three pieces.

In this game no two pieces can be on the same square. The pieces attack in the normal chess fashion, except that an intervening piece does not block the attack of the bishop or rook.

You enter your guess by entering the square's coordinates consecutively without a space, comma, or NEWLINE between them. The first number is across, the second down. This means that the upper left square is 11 (one across, one down), the upper right is 81, the lower left is 18, and the lower right is 88. Pressing NEWLINE enters the guess. Guesses off the chessboard are rejected (so you would get no response for a guess such as 03) but they are tallied in the running total of the number of guesses made.

Each individual square of the chessboard is made up of four character display squares. For the purpose of explanation these numbered 1--4 in the diagram.

1	2
3	4

If no chess piece is at the position guessed, a 0 will appear in square 1 of that position. But if a piece is at the guessed position a B, K, or R will appear in square 1 representing the bishop, knight, or rook respectively. In display squares 2, 3, and 4 a B, K, or R will appear if the corresponding piece is attacking the chosen square.

For example, if you had entered 46, the square 4 across and 6 down on the chessboard might appear as:

O	B
■	R

The 0 means that no piece is on square 46, but the B and R mean that both the bishop and the rook are attacking square 46.

A guess of 35 might give:

B	■
■	■

No piece is attacking the square, but you have found the bishop!

After four guesses—46, 35, 16, 78—the screen might show a display such as Figure 1.

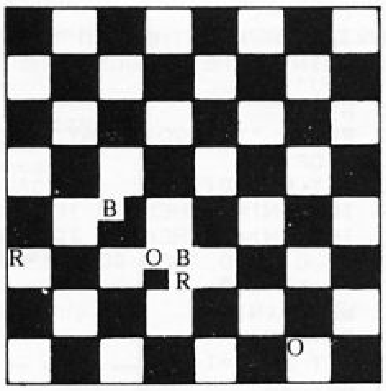


Figure 1.

The program does not tell you when you have found all three pieces. When you believe you have found all the chessmen or have given up, you can enter 0 as your guess. The location of the pieces and the number of guesses you made will appear below the normal display. If after the four moves made in the above example you entered a zero, the following would be added to the display:

```
B35 K81 R16
4 Guess
```

The program uses all of the 1K ZX80 memory. To conserve memory, we used all the one digit line numbers (1-9) and then only the two digit line numbers (10-99). This does not save in the line numbering, but in the GOTO lines we save 5 bytes.

More memory space was saved by switching from two character variable names like B1 and B2 to one character names such as B and C. This saved another 40 bytes.

The computer saves the value of variables in the variable section of its memory. To save more space, we reused the names of some variables: X and Y from the PRINTing routine, and Q from the INPUTting routine. This lets the computer forget information no longer useful and remember new data in its place.

When a variable is first used, it is given a spot in the variable storage and this moves the memory location of the display file. A problem would occur if Q had not been used before line 50 where the start of the display file is found by PEEKing. A memory location for Q would have been made after the PEEKing. The display file is not where your program thinks it is. This would mess up the information POKEd onto the board by the first guess (and could be a problem in similar programs).

POKEing into the display file creates one other problem: it is easy to remove a line delimiter (code 118). This can produce lines longer than 32 characters which crash the program. Unplugging the computer seems to be the only solution. Before running a new program which includes POKEing into the display file, you should save it. This avoids retyping all of the program should it crash. Also beware of a POKEing mistake that changes the program.

Roger and Susan Haar, 19372 Holts East Rd., Martin, OH 43445.

The Hidden Chessmen Program Listing

```
 1 LET B=RND(8)
 2 LET C=RND(8)
 3 LET K=RND(8)
 4 LET L=RND(8)
 5 IF K=B AND L=C THEN GOTO 3
 6 LET R=RND(8)
 7 LET S=RND(8)
 8 IF R=B AND S=C OR (R=K AND S
   =L) THEN GOTO 6
 9 LET N=0
10 FOR X=1 TO 4
12 FOR Y=1 TO 2
14 PRINT "AA##AA##AA##AA##"
16 NEXT Y
18 FOR Y=1 TO 2
20 PRINT "##AA##AA##AA##AA"
21 NEXT Y
22 NEXT X
30 PRINT
32 PRINT "ACR; DN?"
34 PRINT
40 INPUT Q
42 IF Q=0 THEN GOTO 90
43 LET N=N+1
44 LET X=Q/10
46 LET Y=Q-10*X
48 IF X<1 OR Y<1 OR X>8 OR Y>
   8 THEN GOTO 40
50 LET Q=PEEK(19396)+256*PEEK(
   19397)+2*X-1+34*(Y-1)
52 POKE Q,28
54 IF X=B AND Y=C THEN POKE Q,
   39
56 IF X=K AND Y=L THEN POKE Q,
   48
58 IF X=R AND Y=S THEN POKE Q,
   55
60 IF ABS(X-B)=ABS(Y-C) AND NO
   T (X=B) THEN POKE Q+1,39
62 IF ABS(X-K)=1 AND ABS(Y-L)=
   2 OR (ABS(X-K)=2 AND
   ABS(Y-L)=1) THEN POKE Q+17. 48
64 IF (X=R OR Y=S) AND NOT (X=
   R AND Y=S) THEN POKE Q+18, 55
66 GOTO 40
90 PRINT "B#";B;"#";C,"K#";K;"
   #";L,"R#";R;"#";S
92 PRINT N;"#GUESS"
```

Variables List

B,C; K,L; R,S — Chess piece coordinates

X,Y — FOR NEXT variables; later coordinates of guess.

Q — INPUT variable; later location of the guessed chess square in the display file.

N — number of guesses.

Program Summary

Lines 1-8 place the chess pieces.

Line 9 sets guess counter to zero.

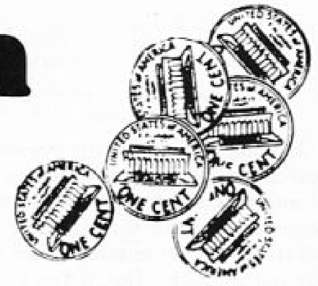
Lines 10-34 print the chess board.

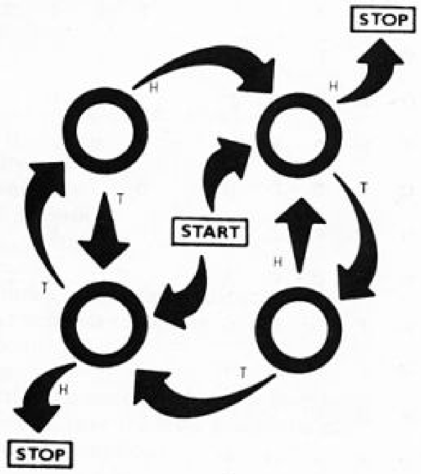
Lines 40-48 input guess and process it.

Lines 50-64 put information about guessed square onto the displayed chessboard.

Line 66 returns to input.

Lines 90-92 print the piece location and number of guesses. ■





"Heads Up Game." Starting with tails in all positions, how many times through to get all four pennies heads up?

4K ROM
2K RAM

keoshyeluncgjhpwsmxreoap
ahtyvpkwqhdxmzpruekfnhw

Create a Word Search Puzzle

aewpnfywvlsyeondqixhurpm
idgaewmbpjdzaugofnepkuyt

Bill McCray

Judging from the many books on word search (or "find a word") puzzles available at any newsstand, these puzzles must be popular. To save some money, you can create your own puzzles rather than buy the puzzle books. But, if you create your own manually, you know where the words are so it is not much fun. There is another way. By using this program in the ZX80 or MicroAce, you can create your own 15- by 11-character word puzzles and have fun solving them or give them to your friends to solve.

The program requires 2K of memory to run and just barely fits into that. So changes to the program that you may want to make must not increase its size if you have only 2K.

When the program is run, the following is displayed.

CREATE A WORD SEARCH PUZZLE

ENTER WORDS OF UP TO 11 LETTERS EACH

FIRST WORD? NULL ENTRY TO END.

The first word is keyed in, followed by NEWLINE. The program responds in one of four ways to each word entered.

1. "word entered"
   NEXT WORD? NULL ENTRY TO END.

The program has successfully embedded the word and wants another.

2. "word entered"
   WORD DOES NOT FIT
   NEXT WORD? NULL ENTRY TO END.

The program was unable to embed the word. Although the puzzle is getting full, another word may fit.

3. "word entered"
   TOO MANY LETTERS
   REENTER

The word entered is longer than 11 letters. Try another.

4. "word entered"
   INVALID CHARACTER
   REENTER

The word entered contains a character other than a letter. Reenter it correctly.

After all words have been entered or the program appears to be unable to embed any more words, a null entry (NEWLINE only) ends the entry and puzzle-construction phase.

The output phase asks the user to select among three options by printing

(S)OLUTION, (P)UZZLE, OR (E)ND?

Entering E and NEWLINE ends the program and returns to the Basic mode. Entering S and NEWLINE prints the solution to the puzzle. P and NEWLINE or any other entry prints the puzzle

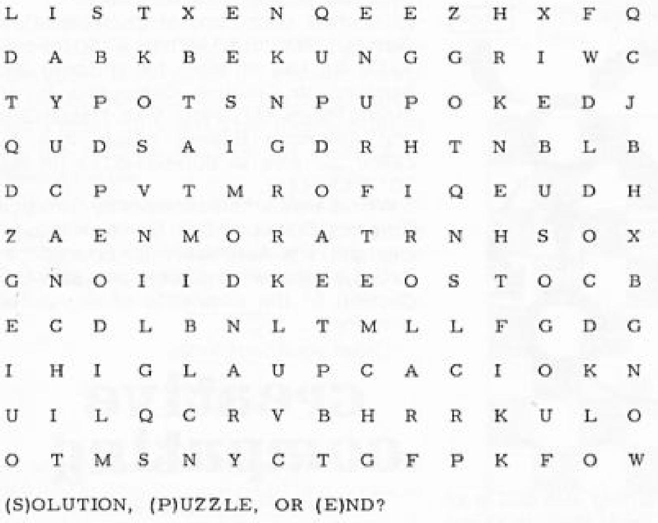
As an example, the following Basic words were entered into the program.

CLEAR
CLS
CONTINUE
DIM
FOR
GOSUB
GOTO
IFTHEN
INPUT
LET
LIST
LOAD
NEW
NEXT
POKE
PRINT
RANDOMISE
REMARK
RETURN
RUN
SAVE
STOP

The puzzle listing generated by P and NEWLINE is seen in Figure 1.

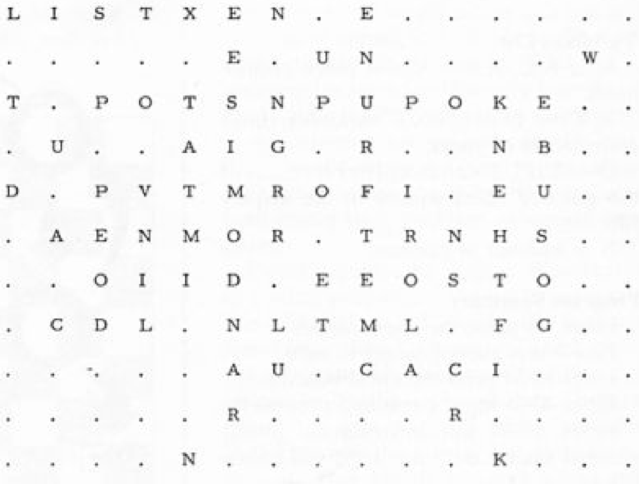
The listing contains the embedded letters and randomly-generated letters in all unused positions.

Bill McCray, 577 Bellcastle Road, Lexington, KY 40505.



```
L   I   S   T   X   E   N   Q   E   E   Z   H   X   F   Q

D   A   B   K   B   E   K   U   N   G   G   R   I   W   C

T   Y   P   O   T   S   N   P   U   P   O   K   E   D   J

Q   U   D   S   A   I   G   D   R   H   T   N   B   L   B

D   C   P   V   T   M   R   O   F   I   Q   E   U   D   H

Z   A   E   N   M   O   R   A   T   R   N   H   S   O   X

G   N   O   I   I   D   K   E   E   O   S   T   O   C   B

E   C   D   L   B   N   L   T   M   L   L   F   G   D   G

I   H   I   G   L   A   U   P   C   A   C   I   O   K   N

U   I   L   Q   C   R   V   B   H   R   R   K   U   L   O

O   T   M   S   N   Y   C   T   G   F   P   K   F   O   W

(S)OLUTION,  (P)UZZLE,  OR  (E)ND?
```

Figure 1.



```
L   I   S   T   X   E   N   .   E   .   .   .   .   .   .

.   .   .   .   .   E   .   U   N   .   .   .   .   W   .

T   .   P   O   T   S   N   P   U   P   O   K   E   .   .

.   U   .   .   A   I   G   .   R   .   .   N   B   .   .

D   .   P   V   T   M   R   O   F   I   .   E   U   .   .

.   A   E   N   M   O   R   .   T   R   N   H   S   .   .

.   .   O   I   I   D   .   E   E   O   S   T   O   .   .

.   C   D   L   .   N   L   T   M   L   .   F   G   .   .

.   .   .   .   .   A   U   .   C   A   C   I   .   .   .

.   .   .   .   .   R   .   .   .   .   R   .   .   .   .

.   .   .   .   N   .   .   .   .   .   .   K   .   .   .
```

Figure 2.

At this point the user can copy the puzzle onto paper. The output options are listed at the bottom for selection. The selected letter does not show on the display when it is entered, but it is still input.

If SOLUTION is selected, the same listing is generated, but periods are substituted for the randomly-generated letters in the unused positions. The solution listing for the puzzle in Figure 1 is shown in Figure 2.

The solution and puzzle may be listed as many times as desired until E is selected. Each time P is selected, the randomly-generated letters change. The embedded letters, of course, are unchanged.

Hints for Users

Long words should be entered first. Since more of the spaces are unoccupied, the chances of successful embedding are improved.

If the program is able to embed all but a small number of the words desired, running the program again may allow the entire list to be embedded, since randomness is used in the embedding process.

Algorithm

A word may be embedded in any of eight orientations, corresponding to the eight points of the compass. One of these directions is chosen randomly. All possible locations for the word in that orientation are checked. The location giving the greatest degree of overlap with the previously positioned words is chosen. If no overlap is possible, the word may be embedded without overlap, if there is space open yet. If the word cannot be placed in the selected orientation, the other seven orientations are investigated in turn in the same manner until embedding is accomplished or is found to be impossible.

Variables

A array	Contents of the puzzle locations.
B array	Letters of the word to be embedded.
A$	General character variable, many uses.
A	Overlap level attained.
B	Overlap counter for position being investigated.
C	Length - 1 of the word to be embedded.
D - G	Horizontal and vertical limits of the possible starting positions for the orientation being investigated.
I - M	General and loop variables.
P	Starting location for the word position giving the overlap level given in A.
Q	Letter displacement for the word position starting in P.
U - V	Horizontal and vertical displacements for the orientation being investigated.
W	Sum of U and V; the total displacement.
X - Y	Horizontal and vertical position of the starting position being investigated.

Program Notes

10-130	Initialize the program and give the first output.
140-180	Get a word and output it.
200-300	Test the word for validity.
310-350	Select an orientation randomly.
360-670	Test a possible embedding position.
360	Initializes the overlap level attained.
380	Changes the orientation.
390-420	Calculate the X and Y limits for the orientation.
430-440	Select a location within the limits randomly.
460-500	Move to the next location.
520-590	See if the word fits here.
580	Increment overlap counter if a matching letter is found.
600-640	If greater overlapping found, save the level and position.
655	Embedding possible in this orientation?
700-760	Embed word.
780-790	Go to get another word.
800-940	Output results.
950-990	Subroutine to change orientation.

Program Listing

```
 10 DIM A(164)
 20 DIM B(10)
 40 RANDOMISE
 50 FOR I=0 TO 164
 60 LET A(I)=0
 70 NEXT I
 80 PRINT " CREATE A WORD SEARCH
    PUZZLE
 90 PRINT
100 PRINT "ENTER WORDS OF UP TO
    11 LETTERS"
110 PRINT "EACH"
120 PRINT
130 LET A$="FIRST"
140 PRINT A$;" WORD? NULL
    ENTRY TO END."
150 INPUT A$
160 CLS
170 IF A$="" THEN GO TO 800
180 PRINT A$
200 FOR C=0 TO 10
210 LET B(C)=CODE(A$)
220 IF B(C)<38 THEN GO TO 290
230 LET A$=TL$(A$)
240 IF A$="" THEN GO TO 310
250 NEXT C
260 PRINT "TOO MANY LETTERS"
270 PRINT "REENTER"
280 GO TO 150
290 PRINT "INVALID CHARACTER"
300 GO TO 270
310 LET U=1
320 LET V=0
330 FOR I=1 TO RND(8)
340 GO SUB 950
350 NEXT I
360 LET A=-1
370 FOR I=1 TO 8
380 GO SUB 950
390 LET D=-C*(U<0)
400 LET E=15+C*(U>0)
410 LET F=-C*(V<0)
420 LET G=11+C*(V>0)
430 LET X=RND(E-D)+D-1
440 LET Y=RND(G-F)+F-1
450 FOR J=1 TO (E-D)*(G-F)
460 LET X=X+1
470 IF X<E THEN GO TO 510
480 LET X=D
490 LET Y=Y+1
500 IF Y=G THEN LET Y=F
510 LET B=0
515 LET M=X+15*Y
520 FOR K=0 TO C
530 LET L=A(M)
540 LET M=M+W
560 IF L=0 THEN GO TO 590
570 IF NOT L=B(K) THEN GO TO 660
580 LET B=B+1
590 NEXT K
600 IF NOT B>A THEN GO TO 660
610 LET A=B
620 LET P=X+15*Y
640 LET O=W
660 NEXT J
665 IF A>-1 THEN GO TO 700
670 NEXT I
680 PRINT "WORD DOES NOT FIT"
690 GO TO 780
700 FOR K=0 TO C
750 LET A(P)=B(K)
755 LET P=P+O
760 NEXT K
780 LET A$="NEXT"
790 GO TO 140
800 PRINT "(S)OLUTION, (P)UZZLE,
    OR (E)ND?"
810 INPUT A$
815 IF A$="E" THEN LIST
820 CLS
830 LET I=0
840 FOR K=0 TO 10
850 FOR J=0 TO 14
860 LET L=A(I)
862 IF L THEN GO TO 880
864 LET L=27
870 IF NOT A$="S" THEN LET
    L=RND(26)+37
880 PRINT CHR$(L);" ";
890 LET I=I+1
900 NEXT J
910 PRINT
920 PRINT
930 NEXT K
940 GO TO 800
950 LET W=(U>-1)*(V>-1)-(U<1)*(V<1)
960 LET V=15*((U<1)*(V>-1)-(U>-1)
    *(V<1))
970 LET U=W
980 LET W=U+V
990 RETURN
```

4K ROM
1K RAM

Two Challenges of Taxman

Austin R. Brown, Jr.

Taxman has been my favorite mathematical computer game since I first discovered it in the Apple Pi program library more than two years ago. (Apple Pi is the Denver area Apple users group.) Taxman is a challenging game to play. With the advent of the ZX80 came a further challenge: could Taxman be adapted from a "large" computer (Apple II) to a really small computer, the Sinclair ZX80?

Was the second challenge met? The answer is a qualified "Yes". Here is a complete program for Taxman, reduced in size from 50 maximum to 30 maximum. Some of the nicities of user interaction are gone, but I believe that ZX80 Taxman still is an interesting, challenging game.

Before looking at the program and its lessons in compacting large programs for the ZX80, let's play the game.

The Game

The computer will lay out the integers from 1 to a maximum which you have entered. You pick one of these numbers; Taxman gets all the remaining factors of that number. If there are no remaining factors of the number chosen, you cannot have it. When there are no remaining factors of any unchosen number, the game is over; and Taxman gets the remaining numbers. If the sum of your chosen numbers is greater than the sum of Taxman, you win.

Let's play a simple game. First enter or LOAD the program, then RUN.

```
TAXMAN
YOU GET NUMBER; I GET FACTORS
```

This is the abbreviated introductory message.

```
HOW MANY(30 MAX)?
```

Here you choose the size of the game; let's choose 12.

```
1 2 3 4 5 6 7 8 9 10 11 12
PICK A NUMBER
```

Let's pick 12.

```
YOU GET 12
I GET 1 2 3 4 6
SCORE- YOU:12 ME:16
5 7 8 9 10 11
PICK A NUMBER
```

Taxman took all the factors of 12, which added up to 16. Now from the remaining numbers let's pick 11.

```
NO FACTOR FOR ME,
TRY AGAIN.
5 7 8 9 10 11
PICK A NUMBER
```

The only factor of 11 is 1. It is already gone. Since Taxman must always get something, we cannot choose 11. Let's try 10.

```
YOU GET 10
I GET 5
SCORE- YOU:22 ME:21
7 8 9 11
DONE
I GET 7 8 9 11
FINAL SCORE- YOU:22 ME:56
```

Since no factors remained for any remaining numbers, the game is over and the Taxman gets the balance, and Taxman wins.

Can Taxman be beaten? Play again with 12. This time start by picking 11, then 4, etc. It is not easy, but you can beat the Taxman.

The Program

Among the factors involved in writing an interactive program are personalization, on-line documentation, and interaction. What does Taxman teach us about these factors in the severely restricted environment of the 1K Sinclair ZX80?

Personalization is that aspect of an interactive program which starts with WHAT IS YOUR NAME?, then goes on to PICK A NUMBER, BOB, encourages with TERRIFIC, BOB!, chastises with COME NOW, BOB, etc. In many programs it is a frill rather than a part of the essence of the program. Out it goes! With less than 1024 bytes at our disposal, we cannot afford the space. But wait until 16K memory arrives; personalization will be back.

On-line documentation consists of the information the program gives us about itself while it is running. Ideally, this should include everything we need to know to run the program; no external manual or other printed instructions should be needed. The realities of a 1K byte machine take us a long way from the ideal. Sometimes, as in *Acey Ducey* (*SYNC* 1:16), no instructions are given. If possible, a program should at least include its name and one line of description or instruction (lines 20-30 of the program).

However, adding text to a ZX80 program can cause additional complications. Most video-oriented personal computers reserve a fixed block of memory for display; the ZX80 does not (*ZX80 Operating Manual*,

Austin R. Brown, Jr., 407 Peery Parkway, Golden, CO 80401.

p. 108). Therefore, if text is added which increases the maximum size of the display of a ZX80 program, the program may bomb with error code 4, even though the program itself will fit in memory. For example, run *Taxman* with 30 numbers, picking successively 24, 30, 14, 27, 26, 22. Now add line 605 PRINT and rerun with the same pattern. There is no longer room for the last few characters in the final display, hence the error ending.

Most interesting or useful programs for personal computers are interactive; they require some response from the user, whether data or decisions. The user must know what is expected when it is time for input; there is nothing as frustrating as an isolated question mark, much less only the cursor leaving you to wonder, "What do I do now?" Give as much information in the questions as possible. Lines 110 and 200 are much clearer than they would be if they said, "SIZE?" and "PICK."

Ideally, any interactive program should maintain complete control over input, so that it is impossible to make an incorrect response. There is not much room for input control in 1K machine. The ZX80 helps us with the LS cursor when a number is expected. We can also check for range, as in lines 130 and 230. But what happens if zero or a negative number had been given for the game size? Would it have been better to cut the maximum size of the game in order to have more complete error checking?

Every program should have good logic and be written in an understandable way. One of the virtues of a tiny computer is that it forces the programmer to learn to write efficient programs if the program is to do anything significant. Efficient logic is not always clear logic. REMarks are important for clarity. However, REMarks take up space. One common solution is to have two versions of a program, one with REMarks for the human and one without for the computer. Note in the listing that all REMarks have line numbers ending in 5. These can be eliminated from the running version of the program if you have a 1K RAM or if you do not want to enter them.

Lines 240-350 give another lesson from Taxman about ZX80 programming. The ZX80 displays output on the TV screen only when all computation is completed. This is usually considered a disadvantage; it can be turned to an advantage when a given display may or may not be wanted. The display of the division of the spoils is always started. If there are no factors for Taxman, line 350 sends control to line 800, where the display is cleared before it is ever seen.

Program Listing

None of the REMarks are to be entered into the ZX80; they are included to show you the logic of the program.

If you use the Syntactic Sum<TM> method to verify that you have entered the program correctly, you must SAVE the program, then NEW, then LOAD before running. Taxman is too large to run with the Syntactic Sum program in memory. ▪

```
  5 REM taxman for ZX80
 10 DIM F(30)
 15 REM introduction
 20 PRINT "TAXMAN"
 30 PRINT "YOU GET NUMBER; I GE
    T FACTORS"
 75 REM initiate
 80 LET B=0
 90 LET T=0
100 PRINT
110 PRINT "HOW MANY(30 MAX)?"
120 INPUT N
130 IF N>30 THEN GO TO 110
140 CLS
150 FOR I=1 TO N
160 LET F(I)=1
170 PRINT I;"#";
180 NEXT I
190 PRINT
195 REM process choice
200 PRINT "PICK A NUMBER"
210 INPUT K
220 CLS
230 IF F(K)=0 OR K>N OR K<2 THE
    N GO TO 850
240 PRINT "YOU GET#";K
250 REM look for factors
255 PRINT "I GET#";
260 LET C=0
270 FOR I=1 TO K/2
280 IF F(I)=0 OR (K/I)*I<K THEN
    GO TO 330
290 LET C=1
300 LET F(I)=0
310 LET T=T+I
320 PRINT I;"#";
330 NEXT I
340 PRINT
350 IF C=0 THEN GO TO 800
360 LET B=B+K
370 LET F(K)=0
430 PRINT
440 PRINT "SCORE- YOU:";B;"#ME
    :";T
450 PRINT
470 FOR I=1 TO N
480 IF F(I)=0 THEN GO TO 500
490 PRINT I;"#";
500 NEXT I
510 PRINT
515 REM check for no more factors
520 FOR I=4 TO N
530 IF F(I)=0 THEN GO TO 590
540 FOR K=2 TO I/2
550 IF NOT F(K)=0 AND (I/K)*K=I
    THEN GO TO 200
580 NEXT K
590 NEXT I
595 REM wrapup
600 PRINT "DONE"
610 PRINT "I GET#";
630 FOR I=4 TO N
640 IF F(I)=0 THEN GO TO 670
650 LET T=T+I
660 PRINT I;"#";
670 NEXT I
680 PRINT
690 PRINT
750 GOTO 999
795 REM no factor found
800 CLS
810 PRINT "NO FACTOR FOR ME,"
820 PRINT "TRY AGAIN."
830 GO TO 470
845 REM illegal choice
850 PRINT "WRONG,"
860 GOTO 820
999 PRINT "FINAL SCORE-##YOU:";
    B;"#ME:";T
    Syntactic Sum=1418
```

resources

Software

- Name change:
  The ZX-GROUP to:
  Heuristics
  25 Shute Path
  Newton, MA 02159

- *Personal Banking System* keeps track of your personal finance and check your bank statements also. £9.95 for cassette and users manual; manual only £2.00. Specify ZX80 or ZX81.
  J. P. Gibbons AIB
  14, Avalon Road,
  Orpington, Kent, BR6 9AX
  United Kingdom

- *Hints and Tips for the ZX81*, £4.95 (plus 75p overseas); *ZX81 Programmers Toolkit* for the 16K machine to do the donkey work when writing your own software, £8.50; *Space Intruders* for 16K, £6.50.
  Hewson Consultants
  7 Grahame Close,
  Blewbury, Oxon OX11 9QE
  United Kingdom

- *Print'N'Plotter*, a tear-off pad of 100 sheets, 11 3/4 x 8 1/4, general planner for ZX Graphics with 2 grids (704 character positions on 1; 2816 pixel coordinates on 2); £3.50 incl. VAT, post, & packing; large orders discounted.
  Dennis Hook
  Butler, Currie & Hook,
  19 Borough High Street,
  London SE1 9SE
  United Kingdom

- *ZX81 Chess*, mc, 6 levels of play, graphic display, 16K, £15.
  A Lawrie (Software)
  68 Bingley Road
  Sunbury-on-Thames
  Middlesex TW16 7RB
  United Kingdom

- *ZX80 Cosmic Dog Fight*, fast action space game for 4K/1K and 16K (8K available Dec. 1), $1.50 pp.
  MATTEX
  P.O. Box 4644
  Shreveport, LA 71104-0644

- *Software Wanted: How and Where to Sell Your Program*, a new guide for those who want to sell programs they have written, intends to bring software authors and marketers together, $25, sold on money-back guarantee basis.

  Battery Lane Publications
  P.O. Box 30214
  301/770-2726

- *Air Traffic Control*, *Invaders*, *Phone Book*, *Date 81* for ZX81 with 16K RAM. Pack 16/81/1 £4.95.
  Control Technology
  39 Gloucester Road
  Gee Cross, Hyde,
  Cheshire SK14 5JG
  Unided Kingdom
  061-368-7558

- ZX81 Software. Send SAE for color catalog.
  CDS Micro Systems
  10 Westfield Close, Tickhill
  Concaster DN11 9LA
  United Kingdom

- *Sixtyfive ZX81 Programs*, 65 1K programs for £4.95. *Beginners Basic Course*, self teach, £10.95.
  Sussex Software
  Wallsend House
  Pevensey Bay
  Sussex
  United Kingdom

Hardware

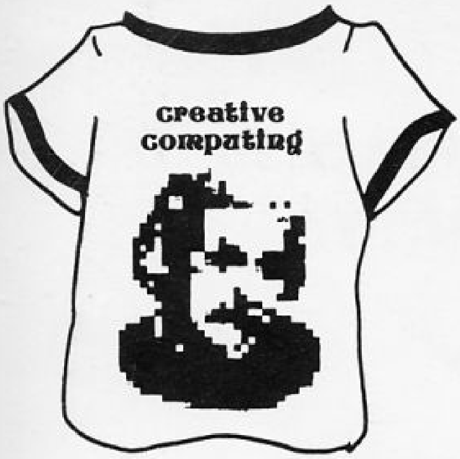
- Self-Reset Power Line Interrupter, disconnects AC power from controlled apparatus, 4 minute time delay, automatic self-reset, helps avoid wide voltage fluctuations, intended for installations unattended for long periods. $185.95.
  Electronic Specialists, Inc.
  171 S. Main St.
  Natick, MA 01760
  617/655-1532

- RAM Packs: 16K, £42.95; 2K, £15.95; 4K, £22.95; 8K, £34.95; Keyboard (13 solder connections to PCB), £27.95. Add £1.00 pp. Specify ZX80 or ZX81.
  dK'tronics
  23 Sussex Road
  Gorleston on Sea
  Great Yarmouth
  Norfolk
  United Kingdom
  (0493) 602453

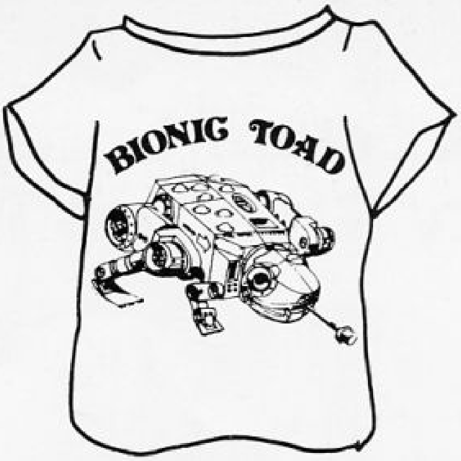
- 24 Line I/O port, controlled using Basic, with suitable interface circuits will control LEDs, motors, relays, lights, sound generators, etc., kit for £14.50, built for £15.95. Add 40p for orders under £10.00; 80p, over £10.00.
  Redditch Electronics
  21 Ferney Hill Avenue
  Redditch
  Worcestershire B97 4RU
  United Kingdom
  (0527) 61240

- Full animated graphics for the ZX80 (no screen flicker); kit form, £12.95 plus VAT; for 8K ROM only.
  Comp Shop, Ltd.
  14 Station Road
  New Barnet
  Hertfordshire, EN5 1QW
  United Kingdom

- FD 81 Keyboard, kit form, £18.95 plus 80p pp; built £24.95
  Fuller Designs, Ltd.
  Sandfield Park East
  Liverpool L12 9HP
  United Kingdom

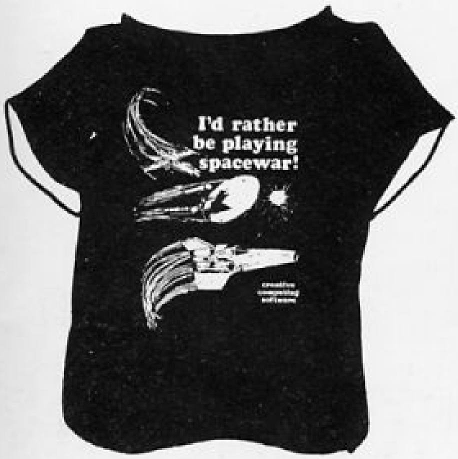


Creative Computing-- Albert Einstein in black on a red denim-look shirt with red neckband and cuffs.



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.

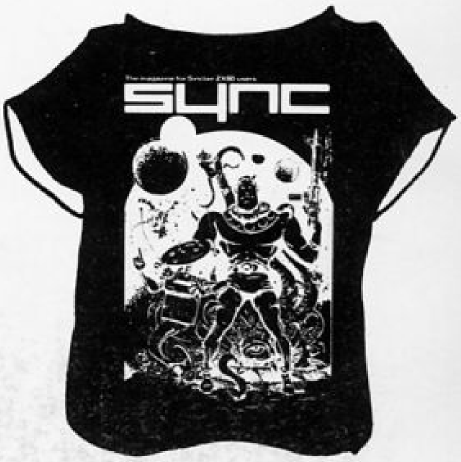


I'd rather be playing spacewar-- black with white spaceships and lettering.

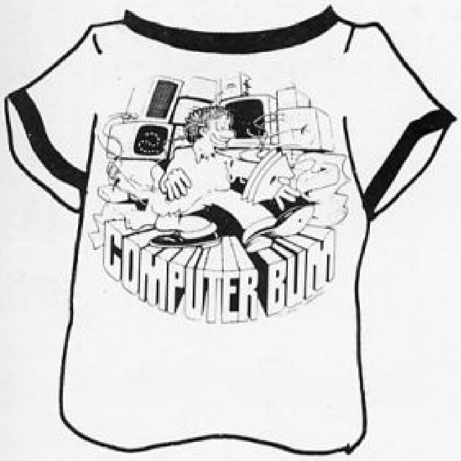
Give your tie a rest!

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.

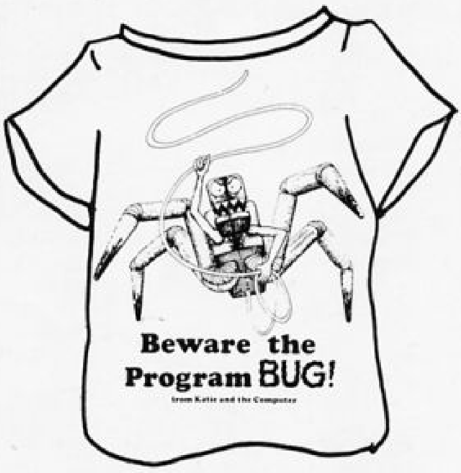
Specify design and size and send payment to Creative Computing, 39 E. Hanover Ave., Morris Plains, NJ 07950. Orders for two or more shirts may be charged to Visa, MasterCard or American Express. Save time and call toll-free **800-631-8112** (in NJ 201-540-0445).



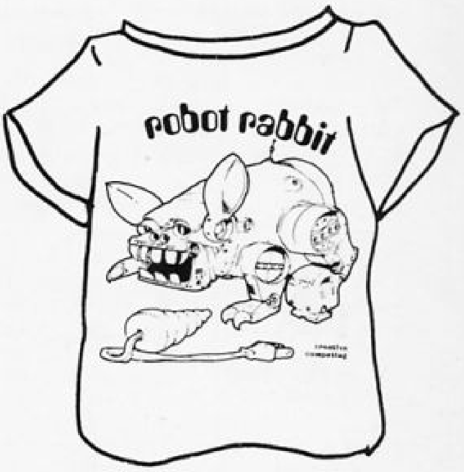
Crash Cursor and Sync from the comic strip in SYNC magazine emblazoned in white on this black shirt.



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.

Plain Talk About Business Computers

Can a Small Computer Really Save You Time?

Time is Money

Theophrastus said time was the most valuable thing a man could spend. Fifteen centuries later Haliburton agreed saying, "we reckon hours and minutes to be dollars and cents." Today, time is more valuable than ever—and more fleeting.

About the only way to gain time is to use it more efficiently and effectively. That's where we come in.

Small Business Computers—by the way, the "small" refers to computers, not to business—will dramatically increase your effectiveness and help save you time and money. How so?

You get flagrantly honest evaluations and reviews of computers and software. We don't just tell you what a program can do; we tell you what it doesn't do, what it does poorly, and what it should do for the price. If advertisers don't like that, we don't want their business, and you're better off without them. Fortunately, most companies appreciate our honesty. In fact, one of our reviewers has gained a reputation because of the many software houses that have incorporated his suggestions into their products. We're proud of that.

Photo courtesy of Alanthus Data Communications Corp.

Plain Talk

Small Business Computers explains the complexity of today's computerized business world without the technical jargon and doubletalk that may have held you back before. In its easily comprehensible "how-to" style, *Small Business Computers* answers your questions while providing the information you need to make some tough decisions. As you select, purchase, and install your computer system, *Small Business Computers* will guide you through each step calmly and comfortably—helping you to evaluate your computer needs and avoid unnecessary pitfalls. As you use your computer, be it mini or micro, *Small Business Computers* will be there to help you do so efficiently and with confidence while informing you of the latest developments and future possibilities of computers in business.

For Example

You have just purchased a mailing list program. Everything is fine until the file has to be sorted by zip code. If the program has that capability, all is well. If not, you have a big problem. If you had just invested a few hours reading *Small Business Computers*, you would have known what functions to look for before buying the program; you would have known how to plan for future needs. That's just one example. Expand this concept into other areas, other programs and systems, and you can see what you get for your investment.

Added Expertise

As the newest member of the Creative Computing family of fine computer publications, *Small Business Computers* will be expanding to offer subscribers more valuable information than ever before. *Creative Computing* editors and contributors will be unleashing their business expertise in *Small Business Computers* through articles, evaluations and applications of particular interest to the business person. *Creative Computing* has a reputation of editorial excellence and integrity built on unbiased, in-depth product evaluations; articles by top thinkers in the field; and pragmatic, innovative applications.

One management consulting firm, for example, used the Shell-Metzner sort described in *Creative*, and saved $3000 a month, and we still receive letters thanking us for the hardhitting, candid, evaluation of word processing printers we published over a year ago, and which, incidentally, cost us several advertisers.

All this knowledge and experience will now be available to business people in *Small Business Computers*.

So, don't let anyone give you that old story about how complicated and difficult computers are. We don't buy that. Our magazine—our whole philosophy—revolves around the sharing of honest information. If you don't know where to start, we'll put you on the right track. If you're already on the road, we'll show you the best route.

For Any Size Business

Whatever your business—manufacturing or banking, retail or research—*Small Business Computers* will increase your efficiency and help save you time and money.

Subscribe today; *Small Business Computers* is the best consultant your business will ever have.

Order Today

To order your subscription to *Small Business Computers* send $12.00 for 1 year (6 issues). If you prefer, call our toll free number **800-631-8112** (in N.J. 201-540-0445) to put your subscription on your Master Card, Visa, or American Express card. Canadian and other foreign surface subscriptions are $18.00 per year and must be pre-paid. We guarantee that you will be completely satisfied or we will refund the remaining portion of your subscription.
Send orders to:

Small
Business Computers
Magazine

39 E. Hanover Ave.
Morris Plains, NJ 07950
800-631-8112
(In NJ 201-540-0445)