People's Computer Company VOL. 5 NO. 5 MAR-APR 77

SUBSCRIPTIONS

People's Computer Company offers software and hardware applications for the novice and intermediate programmer. Applications include tutorials in easy-to-learn-and-use languages, software design, education, recreation, and more.

U.S. Subscriptions
- ☐ $1 single copy
- ☐ $6/yr. (6 issues)
- ☐ $11/2 yrs. (12 issues)
- ☐ Retaining subscription @ $25 ($19 tax deductible)
- ☐ Sustaining subscription @ $100+ ($94+ tax deductible)

Foreign Surface Mail
- ☐ add $4/yr. for Canada
- ☐ add $5/yr. elsewhere

Foreign AIRMAIL
- ☐ add $8/yr. for Canada
- ☐ add $11/yr. for Europe
- ☐ add $14/yr. elsewhere

Back issues, $1 each; indicate Volume and Issue number, how many copies of each.
- Vol. 1, Nos. 1, 2, 3
- Vol. 2, No. 5
- Vol. 3, Nos. 1, 4
- Vol. 4, Nos. 3, 4, 5, 6
- Vol. 5, Nos. 1, 2, 3, 4

Dr. Dobb's Journal, our sister publication, is the home computer user's reference journal for intermediate programmers and heavy hackers. It offers system software, lots of assembly code listings, consumer advocacy info, and realizable fantasies.

U.S. Subscriptions
- ☐ $12/yr. (10 issues)
- ☐ $21/yr. first class/airmail anywhere in U.S.

Foreign Surface Mail
- ☐ add $4/yr. anywhere outside the U.S.

Foreign AIRMAIL
- ☐ add $9/yr. to Canada
- ☐ add $16/yr. to Europe and Pan America
- ☐ add $20/yr. elsewhere

Back issues, $1.50 each; indicate Volume and Issue number, how many copies of each.
- Vol. 1, Nos. 1, 2, 3, 6, 7, 8
- Vol. 2, No. 1
- ☐ all of Volume 1, bound, $13

In Britain:
PACS
c/o John Remizo
142 Grove Lane
Hale, Altrinchan
Cheshire, UK

PCC: £7.50/year
DDJ: £14.00/year

In West Germany:
PACS Gmbh
Frankfurter Strasse 78
D 61 Darmstadt, West Germany

PCC: 30 DM/year
DDJ: 55 DM/year

CONTENTS

EDITOR'S NOTE

Come and get it! Tiny BASIC for beginners, Z–80 source code for PILOT, FORTRAN man, and lots more! Several articles are based on annotated listings: you get a debugged program plus tips on software design, coding, and documenting techniques all in one package. We plan to publish at least one meaty article based on annotated listings in each issue. Usually the listings will be in high-level languages like PILOT or BASIC; lengthy assembly code listings will be found in *Dr. Dobb's Journal*, our sister publication. Do you find the annotated listings useful? Do you think our layouts are easier to read? We await your reactions!

■ ■ ■

RETAINING SUBSCRIBERS
George Bowie III
The Computer Corner, Harriet Shair
Daniel Dick
Bill Godbout Electronics
Mark S. Elgin
Dexter Fletcher
John B. Fried
Lt. Stan Jurgielwicz
John R. Lees, Jr.
John C. Lilly
James Muysenberg
Frank Otsuka
Bernice Pantell
Larry Press
John Rible
Joe Weisbecker

SUSTAINING SUBSCRIBERS
Algorithmics Inc., Bruce Cichowlas
Don L. Behm
Paul, Lori and Tom Calhoun

STAFF

EDITOR
Phyllis Cole
PRODUCTION MANAGER
Meredith Ittner
ARTISTS
Joan Larimore
Ann Miya
TYPISTS
Maria Kent
Susan Winn
CIRCULATION
Beverly Robinson
DRAGON EMERITUS
Bob Albrecht

As ever, thanks to the many many folk who supported our effort in putting this issue together.

People's Computer Company is published six times a year by People's Computer Company, 1010 Doyle St, Menlo Park, CA 94025, a tax-exempt, independent, non-profit corporation. Donations to People's Computer Company are tax deductible. *Postmaster:* Please send Form 3579 to: Box E, Menlo Park, CA 94025. Return postage guaranteed. Application to mail at second-class postage rates is pending at Menlo Park, CA.

Z-80 PILOT
AN EXPERIMENTAL VERSION

Program by Dean Brown Analysis by Marc LeBrun

The recent PCC readership survey indicated that, among other things, there is a fair amount of interest in examining and learning from the programs that are printed. Therefore, in this article, I will provide some analysis and explanation of a 'quick and dirty' implementation of PILOT by Dean Brown of Zilog.

A long-time friend of the PCC family, Dean is a pioneer in the area of humanistic applications of computers in education and has developed and installed systems world-wide. Unlike conventional CAI-type programs with their implicit behaviorism and wrist-slapping approach:

```
    WHAT IS THE CAPITAL OF OUTER MONGOLIA?
>   i don't know
    WRONG, TRY AGAIN.
```

Dean's programs reflect his open, supportive style. User input is indicated by '>'.

```
    WHAT DO TREES HAVE?
>   trunk
    RIGHT, TREES HAVE TRUNK.
    WHAT ELSE DO TREES HAVE?
>   leeves
    ARE YOU SURE TREES HAVE LEEVES?
>   yes
    HMMM . . . THAT'S INTERESTING.
    I DON'T KNOW MUCH ABOUT THAT.
```

Having successfully used various versions of PILOT over the years, Dean decided to brew one up on the Z-80 for himself and friends. The program presented here represents a work in progress, not a finished product, and is offered as a vehicle for learning about both PILOT and assembly level programming and in the hopes of stimulating your interest in developing your *own* PILOT or educational applications systems.

First we present two runs of the same PILOT program (GOLDILOCKS – an old PILOT favorite) showing how the same basic program can 'adapt' to the differing styles of different users. Don't be deceived by the simplicity of the program – it's a powerful tool for getting kids to read and write. Besides, it's fun: even adults familiar with the program enjoy adding outrageous foolishness to this update of the classic children's story.

Here is the Z-80 PILOT program listing. The R statement is for remarks, T for type text, A for ask (for user input) and E for end. Labels are preceded by an asterisk. Additional details and a comparison with the Tiny PILOT presented in the last PCC are provided after the program listing.

Before examining the code for the PILOT interpreter in detail, a few comments seem appropriate. First, in order to read an assembly program at all, you must have some familiarity with the usual conventions in assembly languages (like the use of various columns or fields for different functions, like label, operation code, operands, comments and so on). The most appropriate and rewarding way to learn this stuff is to write your own assembly programs for some particular machine and pick it up as you do so.

Second, you usually need to have a rough idea of the structure of the specific machine being used. This you can get by reading the literature supplied by the manufacturer of the computer. Often you can transfer experience with one machine towards understanding another. The Z-80 is a good example of this. Since it is a sort of souped-up version of the Intel 8080, users of the 8080 should have little trouble following the code, as the function of unfamil-

```
 Once upon a time there were three bears.
 Who do you think they were?
>Mama andPapa andBaby

 Goldilocks didn't like porridge, so she
 found some yogurt in the kitchen. Next she
>ate it all up

 Then the three bears came home. The little bear
 sniffed around and said
>somebody's in my house

 Then the big Daddy Bear chased Goldilocks all
 over the house because she ate it all up.

 Goldilocks hid under the bed.
 Baby Bear hid under the bed.
 They sat on Baby Bear's electric train, and
>went for a ride

 Pretty soon Mama Bear came in yelling
 "Who ate up all my yogurt?"
>oh oh touble coming

 Goldilocks laughed.
 Baby Bear laughed even harder.
 He said "somebody's in my house."
 Mama Bear said
>and she's riding your train

 Then they went on a picnic.
 They packed a basket of peanuts and popcorn and
>jell sandwichwes
 and bubble gum and frisbees and lemonade.

 They decided to go to
>the ocean

 When they got there, the Daddy Bear opened the door
 and all of the jell sandwichwes
 fell out of the car.
 The Baby Bear said "somebody's in my house."
 The Mama Bear said "and she's riding your train."
 The great big Daddy Bear said
>good grief you are very silly

 Goldilocks cried "Let's get out of the ocean.
 Let's go home."
 But the car had a flat tire and they all had to sleep
 in the back seat, and all night long the Daddy Bear grumbled
 "somebody's in my house."
```

```
 Once upon a time there were three bears.
 Who do you think they were?
>the butcher, the baker, and the baker's monster

 Goldilocks didn't like porridge, so she
 found some yogurt in the kitchen. Next she
>pulled her thumb out of the yogurt

 Then the three bears came home. The little bear
 sniffed around and said
>somebody's been thumbing through my monster's yogurt

 Then the big Daddy Bear chased Goldilocks all
 over the house because she pulled her thumb out of the yogurt.

 Goldilocks hid under the bed.
 Baby Bear hid under the bed.
 They sat on Baby Bear's electric train, and
>and through up the switch

 Pretty soon Mama Bear came in yelling
 "Who ate up all my yogurt?"
>Munster said "burp"

 Goldilocks laughed.
 Baby Bear laughed even harder.
 He said "somebody's been thumbing through my monster's yogurt."
 Mama Bear said
>"burp x 2"

 Then they went on a picnic.
 They packed a basket of peanuts and popcorn and
>blueberry yoghurt
 and bubble gum and frisbees and lemonade.

 They decided to go to
>pcc

 When they got there, the Daddy Bear opened the door
 and all of the blueberry yoghurt
 fell out of the car.
 The Baby Bear said "somebody's been thumbing through my monster's yogurt."
 The Mama Bear said ""burp x 2"."
 The great big Daddy Bear said
>o-o-o-o-o-o-o-o-o-o-o-o

 Goldilocks cried "Let's get out of pcc.
 Let's go home."
 But the car had a flat tire and they all had to sleep
 in the back seat, and all night long the Daddy Bear grumbled
 "somebody's been thumbing through my monster's yogurt."
```

iar instructions can often be deduced from their mnemonic, the context, comments or other clues.

Third, it is important to have a clear idea of what the program is supposed to do and how it goes about doing it. These needs should be met by the program documentation, but unfortunately, this is often a neglected area. It is *not* simply that there should be lots of comments in the code, for this only helps explain the local operations to the reader. In addition, there must also be accompanying text which describes what the different routines do, how they interact, what sorts of data they operate on, what assumptions are made and so on. Often this accompaniment takes as much work to produce as writing the original program, but it is also at least as important if someone else is going to read the code. In this case, I have tried to supply some of the background. (Also have a look at the game of ZOT in this issue where I attempt to apply these ideas to higher-level programs as well.)

Keeping the above points in mind we can turn to the actual program: Experimental PILOT for the Z-80. This experimental PILOT is somewhat different from the Tiny PILOT that has been proposed in recent issues of PCC and these differences are worth noting.

Experimental PILOT is a *real program,* not a *specification for a language* as is Tiny PILOT. In particular, it runs on a real computer. The Zilog system it runs on has *floppy discs* on which the PILOT program is assumed to reside. There is of course a *disc operating system,* called ZDOS, which also resides in memory with the PILOT interpreter. The PILOT interpreter can therefore call various *utility routines* in ZDOS to perform certain operations such as input/output to the user's terminal, and other useful things.

The version of PILOT implemented is also different from Tiny PILOT. In particular, there are the following:

1. Experimental PILOT uses no line numbers (changes to the program, which is kept on a disc, are accomplished using an editor). Instead lines *may* be labeled. Labels are an asterisk followed by a simple string such as *LABEL or *Zulch8. Only the first 6 characters of a label are used.
2. There is no U (Use subroutine) statement. Therefore, E (End) always means end of PILOT. Execution of an E statement returns the user to the top level of ZDOS.
3. There is no C (Compute) statement. Therefore, there are no *numeric* variables.

```
R: Goldilocks and you know who; 2/7/77
T: Once upon a time there were three bears.
T: Who do you think they were?
A:
T:
T: Goldilocks didn't like porridge, so she
T: found some yogurt in the kitchen. Next she
*NEXT A:
T:
T: Then the three bears cane home. The little bear
T: sniffed around and said
*SAID A:
T:
T: Then the big Daddy Bear chased Goldilocks all
T: over the house because she \NEXT.
T:
T: Goldilocks hid under the bed.
T: Baby Bear hid under the bed.
T: They sat on Baby Bear's electric train, and
A:
T:
T: Pretty soon Mama Bear came in yelling
T: "Who ate up all my yogurt?"
A:
T:
T: Goldilocks laughed.
T: Baby Bear laughed even harder.
T: He said "\SAID."
T: Mama Bear said
*MAMA A:
T:
T: Then they went on a picnic.
T: They packed a basket of peanuts and popcorn and
*FOOD A:
T: and bubble gum and frisbees and lemonade.
T:
T: They decided to go to
*PICNIC A:
T:
T: When they got there, the Daddy Bear opened the door
T: and all of the \FOOD
T: fell out of the car.
T: The Baby Bear said "\SAID."
T: The Mama Bear said "\MAMA."
T: The great big Daddy Bear said
A:
T:
T: Goldilocks cried "Let's get out of \PICNIC.
T: Let's go home."
T: But the car had a flat tire and they all had to sleep
T: in the back seat, and all night long the Daddy Bear grumbled
T: "\SAID."
E:
```

4. There are no *string* variables either. Strings are not quoted.
5. If an A (Ask) statement is labeled (e.g., *NUG A:) the user's input is saved in a 'variable' of the same name but starting with a backslash (e.g., \NUG) this 'variable' can only be used to type out the user's input again later. The part of the line to the right of the colon in an A statement is always left blank.
6. In addition, A is implemented in the following (peculiar) manner: when a labeled A statement is executed the entire program (a copy of which is kept in memory) is searched for instances of the 'variable' corresponding to the label. These instances are then *replaced* by the input string. For example, in the following program fragment.

```
      T:   HI! WHAT'S YER NAME?
*NAME A:
      T:   WELL MET, \NAME
```

When the A statement is executed and the user types, say TITANIA, the program actually *becomes*

```
      T:   HI! WHAT'S YER NAME?
*NAME A:
      T:   WELL MET, TITANIA
```

This is probably not the best way to implement this. Better, but more difficult would be to implement actual string variables. However, you can still do a lot of interesting things anyway and this will probably be fixed in less experimental versions. Care should be taken, though. For instance, in

```
*ODDPROG   T:   TYPE STRAWBERRIES OR ELSE
*FRUIT     A:
           M:   STRAWBERRIES
         J N:   *ODDPROG
           T:   I SURE LIKE \FRUIT AND CREAM
```

One run might be

```
TYPE STRAWBERRIES OR ELSE
> straw man                                  no match
TYPE STRAWBERRIES OR ELSE
> berry pie                                  no match
TYPE STRAWBERRIES OR ELSE
> strawberries                               match happy now
I SURE LIKE STRAW MAN AND CREAM
```

The last line contains STRAW MAN instead of STRAWBERRIES because the variable marker was replaced the first time the A statement was executed.

7. An R (Remark) statement is used instead of ! for comments.
8. A line without a command letter is taken to be a T statement. Thus :, Y: and N: are taken to be T:, TY: and TN: respectively. (In some PILOTs, such a line is taken to be the *last* statement given.) Also, unknown commands (like X:) are taken to be T's too.
9. There are other miscellaneous unimportant differences, such as no print head positioning, and Tiny PILOT'S 'program mode' is replaced with ZDOS's top level.

We may now examine the program proper. Statements 1 through 26 use calls to ZDOS to do various things necessary to load the PILOT program into memory (starting at WORK). In the course of doing so it invokes the external subroutine GETCHR. The external subroutines are part of ZDOS and are declared in statements 274-280. Their functions are:

NAME	FUNCTION
GETCHR	'rolls' the pointer (HL) into the user's input string (INPUT buffer) to the start of the next 'word', that is, through intervening blanks, CR's, etc.
GET	puts input from the user's terminal into the INPUT buffer
OSLVL	user to return to the operating system level
PUTSTR	sends string in OUTPUT buffer to user's terminal
TTYWR	sends a single character to the terminal
ZDOS	used to call the ZDOS utility routines

Statements 28 through 52 constitute the top level of PILOT. First a colon (:) is found in the section 28-33. Then the pointer into the program (HL) is 'backed up' until it finds a non-blank character (either a command letter or a Y or N) in 34-37. Then in 38-52, the character is tested against all the possibilities and the appropriate branch is taken.

The first section (28-33) calls the useful routine ADVANC located at 202-211. ADVANC 'rolls' the pointer into the PILOT program until it either finds the character in B, which is defined as success and is indicated by a zero-flag, or it finds the character in C, which is interpreted as failure and is indicated by a non-zero flag.

In the case of the initial colon-finding section (29-33) colon is success and an end-of-file character is failure.

The BACKUP section (34-37) works in an obvious manner by decrementing HL until a non-blank is found (which is put into the A register). The various commands are then tested in a skip-chain. (Implementing this as a table search instead might result in more flexibility and tighter code.) An E causes a return to the operating system. The J command invokes the JUMP subroutine at 69-76. JUMP first calls TEXT which moves the pointer (HL) into the PILOT program up to the first non-blank character. This in theory puts HL on the asterisk in the target label name. Next the DE register is made to point at an 8 character buffer called LABEL (293). (LABEL is defined in the code to be only 6 but the next 2 bytes, TALS 12, are free when the program is running. This is done to conserve space.) Then the SEARCH routine (213-257) is called. SEARCH is a complicated area of code which is used by the J, A and M commands in slightly different ways. Rather than going into too much detail I will just describe its overall behavior. In the case of a J command SEARCH scans the entire program until either it finds a string consisting of an asterisk followed by the first six characters of the label or the end of the program. Note in particular the way END? (246-257) takes care of various punctuations. If SEARCH fails JUMP prints a * on the terminal (why not?). If it wins, then HL should be pointing at the labeled line and we continue with that line.

The M statement calls the MATCH routine (92-116). The MATCH routine repeatedly enters the SEARCH routine to see if any of the substrings in the M statement match somewhere in the user's last input. If there is a match, a special match flag (in an alternate register set) is turned on.

The N (and likewise Y) routines work in an interesting manner. If their condition is met they return to BACKUP which causes HL to be backed up to the command letter preceding the N (or Y) which is then processed in the usual manner. If the condition is not set they continue on to the next line.

The R statement continues on in all cases.

Finally, the A command causes the ACCEPT rating (118-210) to be invoked. This is an extremely complicated piece of code which, as mentioned before, replaces all occurances of the specified label with the input string. Complications arise when the substitution causes the line to become too long, etc.

Not counting the appended user PILOT program, the whole interpreter takes less than 500 bytes and represents a typical experimental implementation of a simple language like PILOT. We look forward to seeing *your* programs, either PILOT or other applications, and either finished, or as in this case, experimental versions.

```
                        PILOT  LISTING
 LOC   OBJ CODE    STMT SOURCE STATEMENT

                     1          ; Z-80 PILOT
1C80                 2          ORG 1C80H
                     3
                     4          ; READ TEXT TO WORK SPACE
1C80   219D1E        5          LD HL,WORK       ; SET UP THE ZDOS BUFFER
1C83   E5            6          PUSH HL
1C84   224F1E        7          LD (ADDR),HL
1C87   3EFF          8          LD A,0FFH
1C89   32531E        9          LD (SECTRS),A
1C8C   DF           10          RST GETCHR
1C8D   11541E       11          LD DE,F_NAME
1C90   3E20         12          LD A,' '
1C92   EDA0         13 N_LOOP   LDI               ; LOAD THE FILE NAME
1C94   BE           14          CP (HL)
1C95   20FB         15          JR NZ,N_LOOP
1C97   DF           16          RST GETCHR
1C98   115A1E       17          LD DE,F_TYPE     ; LOAD THE FILE TYPE
1C9B   EDA0         18          LDI
1C9D   214D1E       19          LD HL,DISK
1CA0   360C         20          LD (HL),12       ; OPEN THE DISK FILE
1CA2   CD0013       21          CALL ZDOS
1CA5   3618         22          LD (HL),24       ; READ THE DISK FILE
1CA7   CD0013       23          CALL ZDOS
1CAA   3610         24          LD (HL),16       ; CLOSE THE DISK FILE
1CAC   CD0013       25          CALL ZDOS
1CAF   E1           26          POP HL
                    27
                    28          ; COMMAND TABLE
1CB0   063A         29 PILOT    LD B,':'
1CB2   0EA1         30          LD C,EOF
1CB4   CDD41D       31          CALL ADVANC      ; INCREMENT LINE POINTER
1CB7   C2FE00       32          JP NZ,OSLVL      ; END OF PROGRAM
1CBA   22461E       33          LD (MARKER),HL
1CBD   2B           34 BACKUP   DEC HL
1CBE   7E           35          LD A,(HL)
1CBF   FE20         36          CP ' '
1CC1   28FA         37          JR Z,BACKUP
1CC3   FE45         38          CP 'E'
1CC5   CAFE00       39          JP Z,OSLVL
1CC8   FE4A         40          CP 'J'
1CCA   2826         41          JR Z,JUMP
1CCC   FE4D         42          CP 'M'
1CCE   284F         43          JR Z,MATCH
1CD0   FE4E         44          CP 'N'
1CD2   280E         45          JR Z,NO
1CD4   FE52         46          CP 'R'           ; REMARK
1CD6   2842         47          JR Z,NXTLIN
1CD8   FE59         48          CP 'Y'
1CDA   280E         49          JR Z,YES
1CDC   FE41         50          CP 'A'
1CDE   2869         51          JR Z,ACCEPT      ; UNRECOG & NULL COMMANDS
1CE0   1824         52          JR TYPE          ; ARE TAKEN TO BE "TYPE"
                    53
                    54          ; COMMANDS
1CE2   D9           55 NO       EXX
1CE3   97           56          SUB A
1CE4   B8           57          CP B
1CE5   D9           58          EXX              ; IF THE MATCH FLAG IS OFF
1CE6   28D5         59          JR Z,BACKUP      ; EXECUTE CURRENT COMMAND
1CE8   1830         60          JR NXTLIN        ; ELSE GO TO THE NEXT LINE
                    61
1CEA   D9           62 YES      EXX
1CEB   97           63          SUB A
1CEC   B8           64          CP B
1CED   D9           65          EXX              ; IF THE MATCH FLAG IS ON
1CEE   20CD         66          JR NZ,BACKUP     ; EXECUTE CURRENT COMMAND
1CF0   1828         67          JR NXTLIN        ; ELSE GO TO THE NEXT LINE
                    68
1CF2   CD341E       69 JUMP     CALL TEXT        ; PUT HL ON THE LABEL
1CF5   113E1E       70          LD DE,LABEL      ; AND DE ON LABEL BUFFER
1CF8   3E2A         71          LD A,'*'
1CFA   CDDF1D       72          CALL SEARCH      ; SEEK *LABEL IN THE TEXT
1CFD   28B1         73          JR Z,PILOT
1CFF   3E2A         74          LD A,'*'         ; MESSAGE *
1D01   CD3107       75          CALL TTYWR       ; IF LABEL WAS NOT FOUND
1D04   1814         76          JR NXTLIN
                    77
1D06   2A461E       78 TYPE     LD HL,(MARKER)
1D09   11BA0C       79          LD DE,OUTPUT     ; TARGET IS OUTPUT BUFFER
1D0C   23           80 NXTCHR   INC HL           ; SOURCE IS LINE OF TEXT
1D0D   7E           81          LD A,(HL)
1D0E   FE0D         82          CP ASCICR
1D10   2804         83          JR Z,TYPOUT      ; TRANSFER THE TEXT LINE
1D12   12           84          LD (DE),A
1D13   13           85          INC DE           ; TO THE OUTPUT BUFFER
1D14   18F6         86          JR NXTCHR
1D16   EB           87 TYPOUT   EX DE,HL         ; HL ON CR AT END OF OUTPUT
1D17   CD3C06       88          CALL PUTSTR      ; PRINT THE LINE
1D1A   2A461E       89 NXTLIN   LD HL,(MARKER)
1D1D   1891         90          JR PILOT         ; RETURN FOR NEXT INSTRUCTION
                    91
1D1F   CD341E       92 MATCH    CALL TEXT        ; ROLL UP TO
1D22   224A1E       93 NXTSTR   LD (STRING),HL   ; FIRST CHARACTER IN MATCH
1D25   EB           94          EX DE,HL         ; SPIKE DE
1D26   D9           95          EXX
1D27   97           96          SUB A
1D28   47           97          LD B,A           ; RESET THE MATCH FLAG
1D29   D9           98          EXX
1D2A   214C1E       99          LD HL,RESPNS
1D2D   060D        100          LD B,ASCICR
1D2F   0E2C        101          LD C,','         ; SEEK THE MATCH STRING
1D31   CDF31D      102          CALL FIND        ; IN THE RESPONSE BUFFER
1D34   2006        103          JR NZ,MORE?      ; IF FAIL, SEEK NEXT STRING
1D36   D9          104          EXX
1D37   3C          105          INC A
1D38   47          106          LD B,A           ; SET MATCH FLAG FOR HIT
1D39   D9          107          EXX
1D3A   18DE        108          JR NXTLIN
1D3C   EB          109 MORE?    EX DE,HL
1D3D   7E          110 ADV      LD A,(HL)        ; ROLL UP TO NEXT STRING
1D3E   FE0D        111          CP ASCICR        ; END THE OF MATCH STRINGS
1D40   28D8        112          JR Z,NXTLIN
1D42   FE2C        113          CP ','           ; STRING DELIMITER
1D44   23          114          INC HL
1D45   20F6        115          JR NZ,ADV
1D47   18D9        116          JR NXTSTR
                   117
1D49   FF          118 ACCEPT   RST GET          ; GET INPUT LINE FROM USER
1D4A   21710C      119          LD HL,INPUT      ; SOURCE IS INPUT BUFFER
1D4D   E5          120          PUSH HL
1D4E   063C        121          LD B,'<'
1D50   0E0D        122          LD C,ASCICR
1D52   CDD51D      123          CALL ADVNCE      ; TERMINATE ON <
1D55   E1          124          POP HL
1D56   CAFE00      125          JP Z,OSLVL
1D59   114C1E      126          LD DE,RESPNS     ; TARGET IS RESPONSE BUFFER
1D5C   015100      127          LD BC,LINLIN     ; TRANSFER USER'S INPUT
1D5F   EDB0        128          LDIR             ; TO THE RESPONSE BUFFER
1D61   2A461E      129 LABEL?   LD HL,(MARKER)
1D64   2B          130 LABELA   DEC HL           ; CHECK FOR LABEL
1D65   7E          131          LD A,(HL)
1D66   FE0D        132          CP ASCICR        ; IF THERE IS NO LABEL
1D68   28B0        133          JR Z,NXTLIN      ; RETURN FOR NEXT INSTRUCT
1D6A   FE2A        134          CP '*'           ; LABEL DESIGNATOR
1D6C   20F6        135          JR NZ,LABELA
1D6E   23          136          INC HL           ; SOURCE IS THE LABEL
1D6F   113E1E      137          LD DE,LABEL      ; TARGET IS LABEL BUFFER
1D72   3E5C        138          LD A,'\'         ; LABEL REFERENCE MARK
```

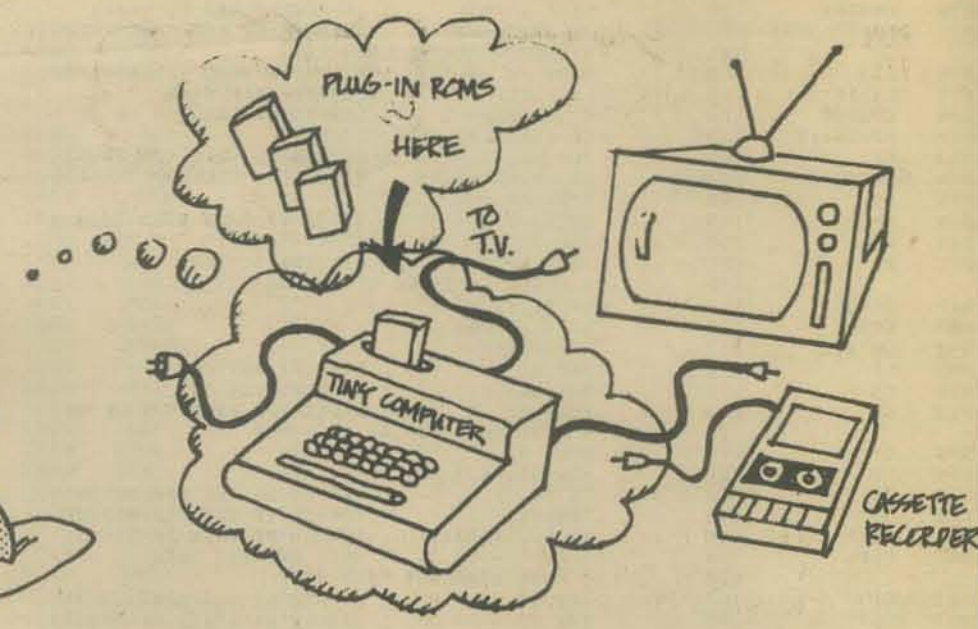
```
1D74  CDDF1D      139          CALL SEARCH      ; IF ACCEPT HAS NO LABEL
1D77  20A1        140          JR NZ,NXTLIN     ; RETURN FOR NEXT INSTRUCT
                  141          ; DELETE \LABEL FROM TEXT
1D79  E5          142          PUSH HL          ; START OF TAIL = \LABEL+1
1D7A  1EA1        143          LD E,EOF         ; END-OF-TAIL MARK
1D7C  CD2A1E      144          CALL SPAN        ; LENGTH OF TAIL IS IN BC
1D7F  ED43441E    145          LD (TALSIZ),BC
1D83  E1          146          POP HL           ; SOURCE IS START OF TAIL
1D84  ED5B481E    147          LD DE,(POINTR)   ; TARGET IS START OF \LABEL
1D88  D5          148          PUSH DE
1D89  EDB0        149          LDIR             ; CLOSE UP TAIL OVER \LABEL
1D8B  1B          150          DEC DE
1D8C  D5          151          PUSH DE          ; CURRENT BOTTOM
                  152          ; MEASURE LENGTH OF RESPONSE
1D8D  214C1E      153          LD HL,RESPNS
1D90  1E0D        154          LD E,ASCICR
1D92  CD2A1E      155          CALL SPAN
1D95  E1          156          POP HL           ; CURRENT BOTTOM
1D96  0B          157          DEC BC           ; ACCOUNT FOR CR
1D97  C5          158          PUSH BC          ; LENGTH OF RESPONSE IN BC
                  159          ; MOVE TAIL OUT BY BC BYTES
1D98  E5          160          PUSH HL
1D99  09          161          ADD HL,BC
1D9A  EB          162          EX DE,HL         ; TARGET IS THE NEW BOTTOM
1D9B  E1          163          POP HL           ; SOURCE IS THE OLD BOTTOM
1D9C  ED4B441E    164          LD BC,(TALSIZ)   ; LENGTH OF TAIL IS IN BC
1DA0  EDB8        165          LDDR             ; MOVE OUT THE TAIL
                  166          ; MOVE RESPONSE INTO TEXT
1DA2  C1          167          POP BC           ; LENGTH OF RESPONSE IN BC
1DA3  D1          168          POP DE           ; TARGET IS START OF INSERT
1DA4  214C1E      169          LD HL,RESPNS     ; SOURCE IS RESPONSE BUFFER
1DA7  EDB0        170          LDIR
                  171          ; TRUNCATE LINE IF TOO LONG
1DA9  1B          172 BACK     DEC DE
1DAA  1A          173          LD A,(DE)
1DAB  FE3A        174          CP ':'           ; CARRY DE BACK TO MARKER
1DAD  20FA        175          JR NZ,BACK
1DAF  015200      176          LD BC,LINLIM+1   ; COUNT UP ONE LINE LENGTH
1DB2  13          177 ROLLUP   INC DE
1DB3  1A          178          LD A,(DE)
1DB4  FE0D        179          CP ASCICR
1DB6  28A9        180          JR Z,LABEL?      ; SUBSTITUTION COMPLETED
1DB8  10F8        181          DJNZ,ROLLUP
1DBA  1B          182 ROLLBK   DEC DE
1DBB  1A          183          LD A,(DE)
1DBC  FE20        184          CP ' '
1DBE  20FA        185          JR NZ,ROLLBK
                  186          ; INSERT CR: IN TEXT AT END OF FIRST LINE
1DC0  3E0D        187          LD A,ASCICR
1DC2  12          188          LD (DE),A        ; SALVAGE TAIL AS :NEW LINE
1DC3  EB          189          EX DE,HL         ; PUT HL ON THE TAIL
1DC4  1EA1        190          LD E,EOF
1DC6  CD2A1E      191          CALL SPAN        ; LENGTH OF TAIL IS IN BC
1DC9  E5          192          PUSH HL          ; SOURCE IS THE OLD BOTTOM
1DCA  23          193          INC HL
1DCB  EB          194          EX DE,HL         ; TARGET IS OLD BOTTOM + 1
1DCC  E1          195          POP HL
1DCD  EDB8        196          LDDR             ; MOVE OUT TEXT BY 1 BYTE
1DCF  3E3A        197          LD A,':'         ; SET A MARKER ON RESIDUUM
1DD1  77          198          LD (HL),A
                  199          ; LOOK FOR \LABEL AGAIN & REPEAT UNTIL EOF
1DD2  188D        200          JR LABEL?
                  201
                  202          ; SUBROUTINES
1DD4  23          203 ADVANC   INC HL
1DD5  7E          204 ADVNCE   LD A,(HL)
1DD6  B9          205          CP C             ; TERMINATOR IN C
1DD7  2804        206          JR Z,NOFIND
1DD9  B8          207          CP B             ; ROLL HL UP TO CHAR IN B
1DDA  20F8        208          JR NZ,ADVANC
1DDC  C9          209          RET
1DDD  3C          210 NOFIND   INC A             ; SET NZ CONDITION FOR FAIL
1DDE  C9          211          RET
                  212
1DDF  12          213 SEARCH   LD (DE),A        ; ENTRY FROM JUMP & ACCEPT
1DE0  ED534A1E    214          LD (STRING),DE
1DE4  13          215          INC DE           ; START LABEL BUFFER WITH
1DE5  010600      216          LD BC,6          ; SPECIAL CHARACTER
1DE8  EDB0        217          LDIR             ; * IF JUMP, \ IF ACCEPT
1DEA  3E20        218          LD A,' '         ; LABEL TO LABEL BUFFER
1DEC  12          219          LD (DE),A        ; ADD BLANK FOR A DELIMITER
1DED  06A1        220          LD B,EOF
1DEF  4F          221          LD C,A           ; SEARCH ENTIRE TEXT
1DF0  219D1E      222          LD HL,WORK       ; FROM TOP TO BOTTOM
1DF3  22481E      223 FIND     LD (POINTR),HL   ; ENTRY FROM MATCH
1DF6  78          224          LD A,B
1DF7  BE          225          CP (HL)
1DF8  281C        226          JR Z,FAIL        ; FAIL
1DFA  ED5B4A1E    227          LD DE,(STRING)
1DFE  1A          228          LD A,(DE)
1DFF  BE          229          CP (HL)
1E00  23          230          INC HL
1E01  20F0        231          JR NZ,FIND       ; FIND MATCH ON FIRST CHAR
1E03  13          232 MATCH?   INC DE            ; THEN CHECK FOR MATCH
1E04  1A          233          LD A,(DE)        ; IN REST OF STRING
1E05  B9          234          CP C
1E06  2810        235          JR Z,END?        ; HIT
1E08  FE0D        236          CP ASCICR
1E0A  280C        237          JR Z,END?        ; HIT
1E0C  BE          238          CP (HL)
1E0D  23          239          INC HL
1E0E  28F3        240          JR Z,MATCH?
1E10  2A481E      241          LD HL,(POINTR)
1E13  23          242          INC HL
1E14  18DD        243          JR FIND
1E16  3C          244 FAIL     INC A            ; SET NZ CONDITION FOR FAIL
1E17  C9          245          RET
1E18  CB78        246 END?     BIT MTCHFL,B     ; TEST IF IN SEARCH
1E1A  C8          247          RET Z            ; RETURN IF IN MATCH
1E1B  7E          248          LD A,(HL)
1E1C  FE20        249          CP ' '           ; PREVENTS CONFUSION
1E1E  C8          250          RET Z
1E1F  FE2C        251          CP ','           ; OF LABELS SUCH AS
1E21  C8          252          RET Z
1E22  FE2E        253          CP '.'           ; ANT & ANTLER
1E24  C8          254          RET Z
1E25  FE0D        255          CP ASCICR
1E27  20CA        256          JR NZ,FIND
1E29  C9          257          RET              ; RETURN Z CONDITION IF HIT
                  258
1E2A  010100      259 SPAN     LD BC,1
1E2D  23          260 COUNT    INC HL
1E2E  03          261          INC BC
1E2F  7E          262          LD A,(HL)        ; RETURN LENGTH BC STRING
1E30  BB          263          CP E
1E31  20FA        264          JR NZ,COUNT
1E33  C9          265          RET
                  266
1E34  2A461E      267 TEXT     LD HL,(MARKER)   ; STARTING AT THE :
1E37  23          268 BLANKS   INC HL
1E38  7E          269          LD A,(HL)        ; ROLL HL UP TO THE FIRST
1E39  FE20        270          CP ' '
1E3B  28FA        271          JR Z,BLANKS      ; WORD IN THE TEXT LINE
1E3D  C9          272          RET
                  273
                  274          ; EXTERNAL SUBROUTINES
                  275 GETCHR   EQU 18H
                  276 GET      EQU 38H
                  277 OSLVL    EQU 00FEH
                  278 PUTSTR   EQU 063CH
                  279 TTYWR    EQU 0731H
                  280 ZDOS     EQU 1300H
                  281
                  282          ; EXTERNAL BUFFERS
                  283 INPUT    EQU 0C71H
                  284 OUTPUT   EQU 0CBAH
                  285
                  286          ; CONSTANTS
                  287 MTCHFL   EQU 7
                  288 LINLIM   EQU 81
                  289 ASCICR   EQU 0DH
                  290 EOF      EQU 0A1H
                  291
                  292          ; ASSIGNED LOCATIONS
1E3E              293 LABEL    DEFS 6
1E44              294 TALSIZ   DEFS 2
1E46              295 MARKER   DEFS 2
1E48              296 POINTR   DEFS 2
1E4A              297 STRING   DEFS 2
1E4C              298 RESPNS   DEFS 1
1E4D              299 DISK     DEFS 1
1E4E  03          300          DEFB 3
1E4F              301 ADDR     DEFS 4
1E53              302 SECTRS   DEFS 1
1E54  20202020    303 F_NAME   DEFM '
1E5A              304 F_TYPE   DEFS 1
1E5B  53          305 F_UNIT   DEFB 'S'
1E5C              306          DEFS LINLIM-16
                  307 WORK     EQU $
```

CROSS REFERENCE
SYMBOL VAL DEFN REFS

```
ACCEPT 1D49  118   51
ADDR   1E4F  301    7
ADV    1DJD  110  115
ADVANC 1DD4  203   31  208
ADVNCE 1DD5  204  123
ASCICR 000D  289   82  100  111  122  132  154  179  187  236  255
BACK   1DA9  172  175
BACKUP 1CBD   34   37   59   66
BLANKS 1E37  268  271
COUNT  1E2D  260  264
DISK   1E4D  299   19
END?   1E18  246  235  237
EOF    00A1  290   30  143  190  220
FAIL   1E16  244  226
FIND   1DF3  223  102  231  243  256
F_NAME 1E54  303   11
F_TYPE 1E5A  304   17
F_UNIT 1E5B  305
GET    0038  276  118
GETCHR 0018  275   10   16
INPUT  0C71  283  119
JUMP   1CF2   69   41
LABEL  1E3E  293   70  137
LABEL? 1D61  129  180  200
LABELA 1D64  130  135
LINLIM 0051  288  127  176  306  306
MARKER 1E46  295   33   78   89  129  267
MATCH  1D1F   92   43
MATCH? 1E03  232  240
MORE?  1D3C  109  103
MTCHFL 0007  287  246
NO     1CE2   55   45
NOFIND 1DDD  210  206
NXTCHR 1D0C   80   86
NXTLIN 1D1A   89   47   60   67   76  108  112  133  140
NXTSTR 1D22   93  116
N-LOOP 1C92   13   15
OSLVL  00FE  277   32   39  125
OUTPUT 0CBA  284   79
PILOT  1CB0   29   73   90
POINTR 1E48  296  147  223  241
PUTSTR 063C  278   88
RESPNS 1E4C  298   99  126  153  169
ROLLBK 1DBA  182  185
ROLLUP 1DB2  177  181
SEARCH 1DDF  213   72  139
SECTRS 1E53  302    9
SPAN   1E2A  259  144  155  191
STRING 1E4A  297   93  214  227
TALSIZ 1E44  294  145  164
TEXT   1E34  267   69   92
TTYWR  0731  279   75
TYPE   1D06   78   52
TYPOUT 1D16   87   83
WORK   1E9D  307    5  222
YES    1CEA   62   49
ZDOS   1300  280   21   23   25
```

MORE TINY BASIC

by The Dragon



(continued from PCC, Volume 5, Number 4, January 1977)

Last time . . . you may recall . . . we had just unpacked our Tiny Home/School computer, attached the TV cable to our TV set, plugged in the Tiny BASIC ROM, and learned some simple stuff about Tiny BASIC. Here is a brief replay of our efforts to compute the value of N^2 for four values of N (N=23, 37, 53 and 88).

```
. We type           CLEAR
. Computer types    I'M CLEAR

. We type           10 LET N=23
                    20 PRINT N*N←('cause N^2=N*N)
                    RUN

. Computer types    529          ←23^2

. We type           10 LET N=37
                    RUN

. Computer types    1369         ←37^2

. We type           10 LET N=53
                    RUN

. Computer types    2809         ←53^2

. We type           10 LET N=88
                    RUN

. Computer types    7744         ←88^2

                  ↗ ▮
                    Here is the cursor.  What next?
```

A new type of statement, called the INPUT statement, will help us feed numbers into box N.

```
. We type           CLEAR
. Computer types    I'M CLEAR
                                  This is an INPUT statement
. We type           10 INPUT N ↙
                    20 PRINT N*N
                    ▮
```

We have entered a two-statement program, including our new INPUT statement. Let's RUN it.

```
. We type           RUN
. Computer types    ▮
```

The computer turned on the cursor. The computer wants something . . . it wants us to INPUT a number for N.

```
. We type           23
. Computer types    529
                    ▮
```

The statement: 10 INPUT N

tells the computer to turn on the cursor and wait for someone to type a numerical value for N.

So . . . if we type a value for N, the computer puts our value into N and goes on to the next statement. In the above program, the computer moves on to Line 20, computes the value of N*N, and prints it, then stops.

Let's RUN the program again, three more times, for N=37, 53 and 88.

```
RUN      ↙ N
37       ↙ N^2
1369

RUN      ↙ N
53       ↙ N^2
2809

RUN      ↙ N
88       ↙ N^2
7744

▮   and so on . . .
```

Well, just to remind you that Tiny BASIC works with *integers* from –32767 to 32767, we'll RUN it three more times.

```
RUN
40000
MY NUMBER RANGE IS -32767 TO 32767

▮ ←— You get another chance!
```

If you type a number that is *not* an integer from –32767 to 32767, the computer politely tells you its number range and gives you another chance. Here is another example.

```
RUN
3.14
PLEASE TYPE AN INTEGER

▮ ←— Again, you get another chance.
```

And, if you type something other than a number . . .

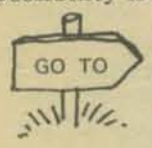
```
RUN                          We typed
TELL ME A STORY
PLEASE TYPE A NUMBER

■
```

Well, just how *do* we get out of this INPUT thing so we can write another program?

On some computers, hold CTRL key down and press C .

On some computers, press BREAK two or more times.

[Actually, we haven't really decided how to do this for the Tiny Home/School computer, but soon we will decide, and tell you. One possibility is to permit the user to type STOP . . . what would you like?]



Here again is our program to type squares of numbers, but with something added.

```
10 INPUT N
20 PRINT N*N
30 GO TO 10
```

Another new statement called GO TO. It tells the computer to "GO TO LINE 10" and continue from there.

Hmmm . . . we assume Lines 10 and 20 are still stored in memory. [Of course, if you have typed CLEAR they won't be . . . or, if you have turned off the computer and turned it back on, Lines 10 and 20 will be gone.]

So . . . just to be sure, we type LIST and press RETURN.

```
. We type              LIST

. Computer types       10 INPUT N      Yup, here is the old
                       20 PRINT N*N        program.
                       ■
```

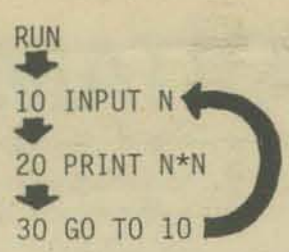
Now let's add our GO TO 10 statement and then LIST the program again.

```
. We type              30 GO TO 10
                       LIST

. Computer types       10 INPUT N
                       20 PRINT N*N   Here is the GO TO, in
                       30 GO TO 10    its proper place, in
                       ■              line number order.
```

Now we will RUN the modified program and see just what this GO TO does.

```
RUN
23        The cursor blinked - we typed 23.
529       The computer typed 529.
37        The cursor blinked - we typed 37.
1369      The computer typed 1369.
53        The cursor blinked - we typed 53.
2809      The computer typed 2809.
88        The cursor blinked - we typed 88.
7744      The computer typed 7744.
■         And so on . . .
```

Remember, Tiny BASIC statements are executed in *line number order* . . . unless a GO TO breaks the order. In the preceding program, the statements are done in the order shown below . . . follow the arrows.



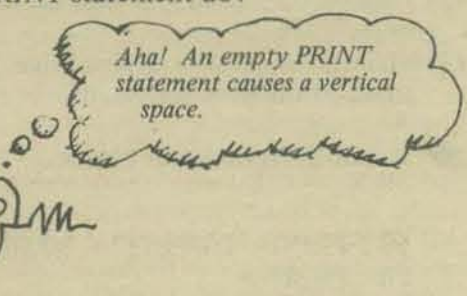
Around and around and around and . . . unless, of course, you stop the computer.

Look at the RUN of the above program. Crowded, isn't it? We will add another PRINT statement *between* Lines 20 and 30, then LIST the new program.

```
. We type              25 PRINT ←— An "empty" PRINT statement
                       LIST
. Computer types       10 INPUT N
                       20 PRINT N*N
                       25 PRINT ←— Here it is, between Line 20
                       30 GO TO 10        and Line 30.
                       ■
```

Now you know why we number lines 10, 20, 30, etc., instead of 1, 2, 3, and so on. That fine habit gives us room to insert a new line between two old lines! But what does an "empty" PRINT statement do?

```
RUN
23
529

37
1369

53
1369

88
7744

■
```



An empty PRINT statement prints an empty line.

Instead of a screen full of numbers, wouldn't you rather have numbers identified by names?

Instead of

```
RUN
23
529

37
1369

■    and so on . . .
```

Wouldn't you prefer the following?

```
RUN
COMPUTE N SQUARED

WHAT IS N?23
N SQUARED=529

WHAT IS N?37
N SQUARED=1369

WHAT IS N?■      and so on . . .
```

Computers are much easier for people to use if we use words to tell the user what is wanted and words to label information typed by the computer. Follow along as we write a program to identify desired INPUT numbers and results printed by the computer.

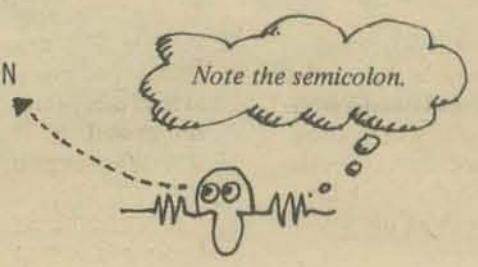
Here are the first two statements

```
10  PRINT "COMPUTE N SQUARED"
20  PRINT
```

Line 10 causes the computer to print the message (string) COMPUTE N SQUARED on the screen. Line 20 is an "empty" PRINT statement and simply causes a vertical space, or "empty line" to be printed.

Next, a very fancy INPUT statement.

```
30  INPUT "WHAT IS N?";N
```



This statement tells the computer:

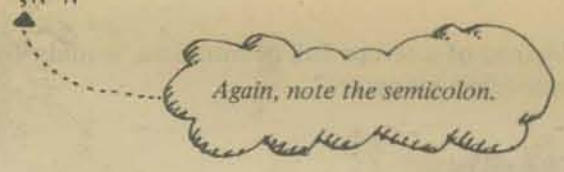
(1) Print the string WHAT IS N? on the screen.
(2) Turn on the cursor and wait for someone to type a numerical value for N.
(3) When someone *does* type a value, put this value in box N, and go on to the next statement.

So far, our program looks like this:

```
10  PRINT "COMPUTE N SQUARED"
20  PRINT
30  INPUT "WHAT IS N?";N
```

At this point, we assume the value of N is available for computing N*N. So, let's do it.

```
40  PRINT "N SQUARED=";N*N
```



This statement tells the computer:
(1) Print the string N SQUARED= on the screen.
(2) Compute the value of N*N and print it on the screen.

One more statement!

```
50 GO TO 20
```

Now let's put everything together, store the entire program, and RUN it.

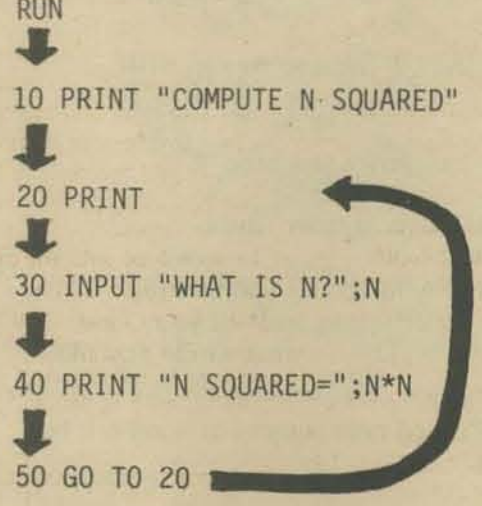
```
CLEAR
I'M CLEAR
10 PRINT "COMPUTE N SQUARED"
20 PRINT
30 INPUT "WHAT IS N?";N
40 PRINT "N SQUARED=";N*N
50 GO TO 20
```

We CLEAR (erase) any old program and type in the new program.

Then RUN it.

```
RUN
COMPUTE N SQUARED

WHAT IS N?23
N SQUARED=529

WHAT IS N?37
N SQUARED=1369

WHAT IS N?▮      and so on, until someone tells
                  the computer to stop.
```

Got it? Just to make sure, follow the arrows.



Your turn.

1. Write a program to do addition, as shown by the following RUN.

```
RUN

IF YOU WILL ENTER VALUES FOR A AND B,
I WILL PRINT THE VALUE OF A + B.

A = ? 7
B = ? 5
A + B = 12

A = ? 3
B = ? -25
A + B = -22

A = ?     . . . and so on.
```

2. Now modify your program (above) so that it RUNs like this:

```
RUN

 IF YOU WILL ENTER VALUES FOR A AND B,
 I WILL PRINT THE VALUES OF A + B, A - B, A * B, AND A/B


A = ? 7
B = ? 5
A + B = 12
A - B = 2
A * B = 35
A/B = 1

A = ?     . . . and so on.
```

NEXT TIME. More about PRINT and INPUT and, introducing, random numbers, the IF statement and computer games in Tiny BASIC.

COMMUNITY INFORMATION SYSTEMS

Reprinted with permission from the October, 1976 Northwest Computer Club Newsletter (P.O. Box 242, Renton, WA 98055).

Communities are defined by communications between people. Just as a house become a home when residents get to know each other, a bunch of people become a community when they exchange information. Traditionally, neighborhoods relied on close blood relationships, a united response to natural disasters, and gossiping between old friends to maintain a community spirit. Today, family ties are weaker, big government handles our problems, and people move too frequently to form many long term friendships. New methods of community communication are needed.

The community memory is an approach to this problem. Members put in short messages, or 'items', into the group mind, along with keywords describing the message. The item may be just 'I am here; I'm interested in X'; it can function as a classified ad, like selling a car or finding a babysitter; it can be a piece of dialog in a community discussion; there are many ways people use such systems. A computerized data base program functions well as a community memory, although keysort cards provide a low cost alternative.

The first computer community memory experiment occurred in the San Francisco area several years ago. Resource One (1380 Howard St., San Francisco CA 94103) is a non-profit group with a donated XDS-940 timesharing computer system. They put five terminals in the Bay Area running a simple keyword retrieval system. The ideas was a success – musicians found new band members using a terminal in a Berkeley record shop, a multiple-author story based loosely on Firesign Theater evolved, cars were sold and babysitters found. The project finally died due to lack of money.

The Community Information Centre of Vancouver uses a keysort card system to distribute information about government services to the public; it's not a community memory since the data only goes one way. However, they spawned a group called Infact, which did set up a public terminal for two-way messages. Infact folded with the progressive Bennett administration in British Columbia, again due to lack of funding. Both Infact and Resource One allowed users to create their own keywords. This allowed greater flexibility, but generated keyword synonyms (such as 'pottery' and 'ceramics') which lead to confusion.

A self-supporting community memory has operated for some time, serving the Pacific Northwest region on a mail order basis. ComNet uses keysort cards to store information about members, and generate directories by keywords such as 'energy', 'food and farming', and so on. They hope to use a computer for this process soon, as the 5,000 keysort cards are getting unwieldy. ComNet is at Box 5599, Seattle, WA 98105; send a self-addressed, stamped envelope (SASE) for more information.

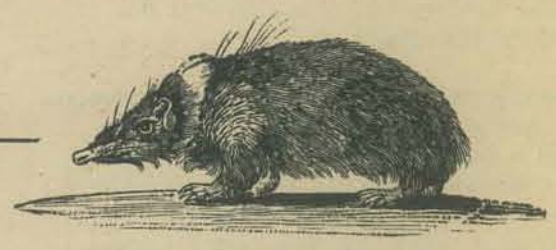
On a neighborhood basis, Peter and Trudy Jounson-Lenz have used their terminal and donated computer time to process a block-grant questionnaire for Lake Oswego, Oregon. Neighbors drop in to search the data base for nearby people with skills to trade or common interests.

The APL program they developed is very flexible in generating reports for their community. A description of the program is available for $1 and a SASE (695 5th Ave., Lake Oswego, Oregon 97034). Steve Johnson of 'Rain' magazine (2270 NW Irving, Portland, OR 97210) is developing a neighborhood information and skills exchange for the Northwest Portland area, which may use Peter and Trudy's program.

Brian Livingston is developing the Cascadian Regional Library, using keysort cards now, and later a computer. It will keep regional information about many topics, such as land use, health, and media, typeset the information and print it. More information is available from CAREL Box 1255, Eugene OR 97401.

A new community memory, the Cruncher, is undergoing final development in the Bay Area. Designed by Lee Felsenstein (who did the Processor Technology VDM and the M & R Enterprises Pennywhistle Modem), the system operates out of a van, using an LSI-11 based keyword retrieval system operating in batch mode. Typically, the van will set up at a central meeting place in a rural area, set up several input terminals, and receive items from community members. Later in the day, the system 'crunches' the data, producing directories organized by keyword. Finally, these directories are given to the community for further distribution. This batch processing mode is cheaper than interactive access to the information, since a large, high-speed disk is not needed. The list of keywords is maintained by a 'gatekeeper' or librarian, working closely with the folks using the data. More information is available from the Journal of Community Communications, 1807 Delaware St., Berkeley, CA 94703 (include $.24 postage).

WHY I HATE MY COMPUTER WHEN IT SPEAKS IN BASIC



BY JAMES W. GARSON

BASIC is by far the most widely used language for writing games, simulations and tutorial programs. You might hope that BASIC could serve as the much needed standard for developing and exchanging this sort of material. But as any anyone who has tried will tell you, there is no such thing as a standard BASIC. Even if there were, BASIC has a number of "features" which make it a particularly poor language outside the realm of mathematical calculation. Many of the problems I want to talk about are not so much the fault of the form of the language, but of the decisions made in the compiler or interpreter and operating system. I don't see how to separate the problems with BASIC neatly into such categories, and I won't attempt to. I am aware that many of the troubles I mention can be avoided in some versions of BASIC, yet they are common enough that BASIC needs a thorough overhaul if it is ever to serve as a convenient standard.

One of the most common myths in the computer community seems to be that BASIC is a good language because it will run on any computer which has BASIC software, and almost every computer does. Those who know a little better say that a program written in BASIC can be expected to run on any machine with "light editing". Neither claim is very accurate. The variations in BASIC affect the very techniques of programming which can be used, not just the spelling and syntax of commands. For instance, there is wide variation among versions of BASIC about which kinds of variables can be subscripted, and how how many subscripts each can bear. Even worse, there are wide differences and often very strict limitations on the word-lengths of the variables, and very little agreement on how and whether the programmer can define functions and subroutines.

The trouble is that these variations not only make learning to use a new version of BASIC more difficult, they make it impossible to run programs written on one computer in a new environment without radical surgery. It is true that if you restrict yourself to the most basic BASIC commands, and if you do not make use of subscripts, string-variables, subroutines, defined functions, and so forth, it is possible to produce programs which run in most places with modest editing. But this lowest denominator BASIC is so impoverished that only the most simple-minded programs can be written in it with any efficiency. So many handy features of BASIC are unavailable that it would almost be better to program in Assembler.

Unfortunately, there is very poor information available on the quirks of the various BASICs. This may help explain why so many people are unrealistically optimistic about how well their programs will run on another computer. The information one needs is neatly scattered through scores of programming manuals, and it never appears where one would expect. Furthermore, there is very poor communication among programmers about their trials and tribulations with BASIC. They tend to suffer in silence and develop their own sets of techniques for overcoming the limitations with their computers. These methods are often quite clever, but they are almost guaranteed not to work in a system that runs another version of BASIC.

One of the major obstacles to writing good programs in BASIC is that BASIC forces a division between numbers and text. In this, it displays its historical roots in FORTRAN, a language designed for calculation in a research setting. In that environment, text was used only to label numbers. Virtually no processing of letters, words, or sentences was possible, since the emphasis was on mathematical operations.

As a result, the weaker versions of BASIC are poor at handling text, either because string variables can contain at most six characters or because string variables cannot be subscripted. This exerts strong pressure on programmers to write programs which expect only numerical input. You may be familiar with the annoying message ("Type 1 for YES, 0 for NO") which results.

BASIC's division between text and numerals leads to many many awkward problems even in systems specially designed for educational uses of the computer, such as the EDU – BASICs for the PDP– 8. When the program calls for the user to input a number, and anything else is typed, the program aborts and a cryptic error message appears: ERROR 34 LINE 510, or some such. This is quite frustrating to students and other users, especially when the program is a game or simulation. It is rarely possible to continue from the point at which the abort occurred, and never when random numbers are used to simulate random events . The whole program has to be started over from the the beginning. These difficulties come from a system design that treats input commands and read commands on the same footing. It is a good idea to send an error message to the programmer when the wrong kind of data is sent to a READ, and it is a good idea to abort the program then so that the programmer can fix the bug. But when an INPUT command is carried out, the person at the terminal will be a user, not a programmer, yet exactly the same thing happens. This is just one example of the orientation towards programmers, not people, which characterizes much of the design of BASIC systems.

The sad thing is that entering a non-number when the computer expects one is a common occurrence, and programs bomb out constantly on these systems, especially when children or poor typists are at the terminal. I am an excellent typist (or so I like to think) and I tend to be pretty good at following directions, yet my first experience with such a system was disastrous. The typewriter I am used to does not have keys for zero and one, so I have the habit of using the *oh* and *el* keys instead. The confusion between these number-letter look-alikes is a constant problem especially for beginners, and my deeply ingrained habits left me in constant frustration. Another common source of trouble is that users often push the return key before any number is entered. They forget that they haven't entered a number yet, or the terminal may be out of adjustment so that the computer sees two returns when the user only typed one. Again, since a number was not entered when the computer was expecting one, the program bombs.

Another problem involves commas. It is natural for most people to use commas in writing large numbers. But this is fatal at the terminal. The comma is used in BASIC to separate two entries, so 2,000 (for example) is read as 2 followed by the separate entry 000. this can also cause a program to abort when it expects to see one number, not two.

Now the obvious answer to all these problems is to write programs that inspect the user's input, check for input errors, and ask to have the errors retyped or simply fix the mistakes. The trouble is that such programs cannot be be written in the weaker versions of BASIC. The decision about what will happen when the user makes these common mistakes has been taken out of the hands of BASIC programmers; it has been written into the compiler or operating system, and so cannot be changed by them.

Of course, there are many BASIC systems which do not have these flaws. In some, for instance, the program continues when data is sent in the wrong form and a polite message such as DATA INCORRECT FORM: PLEASE RETYPE appears. Still, the programmer does not often have control over what the message will say. As a result he may not be able to write a program which inspects the input and sends such messages as, REMEMBER THERE IS IS A DIFFERENCE BETWEEN 0 AND Ø, or PLEASE DON'T PUT COMMAS IN YOUR NUMBERS: THEY CONFUSE ME, or which simply corrects the errors. Although smaller children can read the standard message and make some sense of it, even adults are not likely to know just *why* their entry was not in the correct form.

BASIC is not the only language which suffers from this sort of problem. Alfred Bork complains of a similar problem in using APL for writing tutorial programs. When a student does something unexpected, an error message for the programmer is sent to the terminal, in spite of the fact fact that it is a student, not a programmer, who will see it. Luckily the program does not terminate, but still the student is bound to be mystified, and not likely to have a clue about what to do next to get things going again. This is a feature of APL which cannot be changed without changing the compiler; no cures for it are to be found by programming in APL.

There *are* ways to overcome most of these difficulties with with BASIC in many of its versions. But this is the area where the most variation between versions of BASIC lies; fixes that work on one system cannot be used in other systems. If lowest common denominator systems of BASIC are used in order to solve the problem of transporting programs to other computers, none of them can be employed; the user has to learn all about what the computer expects, and avoid all errors. This can make interacting with the computer extremely slow, intimidating, and frustrating, especially for children.

Even when a program is designed to run on a single computer, it generally takes a lot of talent and effort to overcome these problems, if they are curable at all. When a fix is invented, the same piece of code has to be incorporated in all programs that are written. Not only is this boring, but it may make some programs too large or slow, especially if they are run on a mini or a time sharing system.

Editorial note: We are going to put lots of energy into creating Tiny Languages for Tiny Kids, so don't wait! Get your ideas in NOW to help us include them in designs for Tiny Languages. One nice thing about TINY Languages: they take a lot less time to get together and try them —— we find out we goofed MUCH sooner, and we can change them quickly!

If computers are going to be used by people, everything in our power should be done to make working at the terminal natural and easy. We do not need to train people to accomodate themselves to the poor design of computer systems, we need to redesign our systems to encourage programmers to accomodate people.

Unfortunately, we are caught between the need for standardization which a popular language like BASIC can provide, and the sad fact that BASIC is not well designed for writing games, simulations, CAI or other programs to be run by real people. I think we have to start looking at programming languages in a brand new way. BASIC, and other languages, save the programmer from the terrors of machine code by providing packages which handle input, output, and bookkeeping for variables which are tedious to code directly. The cost is that decisions are made about how these things will happen which programmers can no longer control, or which they have to work around.

The problem is that we are trying to standardize too much at once; an entire body of decisions has to be made in designing a language and operating system, decisions which are supposed to satisfy everybody. The language has to be taken all in one piece, or not all, and it is designed by a group of "experts" whose orientation is more likely to lean towards answers which (they think) are good for research or business, where it is assumed that ordinary folks never see a terminal. This is an unfortunate attitude because it is so short-sighted. Computers will sell better, and find many more applications when it is easy for people to interact with them. Furthermore, certain idiotic features frustrate professional computer people as easily as they do laymen.

What we need is a way to design languages which give people choices between modular chunks of system software, and to have modules available which match people's needs. If most people like to put commas in their numbers, then there ought to be a BASIC that doesn't freak out when they do. One simple solution might be to put some of the alternatives in the same software, and to let programmers set input options for each program that they write. In the case of the comma problem, or the zero-oh confusion, programmers could issue commands in their programs which ask the system to convert ASCII code from the terminal (or EBCDIC if you are so unfortunate as to have it) into an "appropriate" input to the standard BASIC interpreter, oh would be converted to zero, el to one, comma to no signal, and some other handy delimiter to comma. Providing such an option shouldn't tax software writers unduly.

While we are at it, a whole body of handy modules could be designed. There could be a question answering routine, all written in nice fast machine code, which would convert affirmative responses (and some of their common misspellings) to 1, and negative responses to 0, or a routine to check whether a string of characters is a numeral, or one which sends well-thought out error messages when a user makes input mistakes, or anything else that programmers of games, simulations and educational materials seem to use a lot. As it is, each programmer ends up "reinventing the wheel" because all the compiler hands him is a pile of beautifully machined steel blocks. What we need is to develop effective avenues of communication between programmers from all walks of life and software builders about what options are liable to be popular. These options can be either available in a single language-system, or if the system software gets too long, in a family of systems.

In the latter case, software buyers could get a "frame" and a set of modules which fit their specific needs. We might think of the frame as similar to a LISP interpreter, and of the modules as sets of pre-programmed LISP functions.

But wouldn't that simply end up proliferating a whole crowd of new languages? Didn't I begin this by pointing out the need for standardization? If everybody writes with a tailor-made language, then nobody's program will *ever* run anywhere else. But that doesn't follow. though the number of possible languages would be high, the number of modules would be fairly low, since unpopular ones would be dropped by the wayside. Since all modules would fit the frame, all I would need to run your program is the modules you used in writing it. If I didn't happen to have one of your modules I would be out of luck, but then we would both know that your program is not compatible in my environment (information which is scarce nowadays) since we would both know what modules you used, and we would also know exactly what I have to do to fix things up. In fact, you could send me the modules I don't happen to have right along with your program.

By dickering with modules, instead of entire software systems, we can rewrite just the modifications we need without radical overhauling. This way, the design and overhaul of language-systems could become an organic process which responds to the programmer's special and changing needs, while still allowing enough standardization for the easy transportation of programs.

I am sure there are many people out there who think this is a total fantasy. Perhaps part of their trouble is that they can't imagine a world where programmers get a chance to get their hands into the design of system ware, or perhaps they are afraid of a world where small pieces of software are freely exchanged, rather than owned and controlled by computer companies. But when people have no control over what they get, it is easy for computer companies to sell software which makes life difficult for programmers and users. If people get a chance to choose the features they want, a lot of miserable choices which now grace computer language systems would have to change, and companies would have to start competing at making better choices.

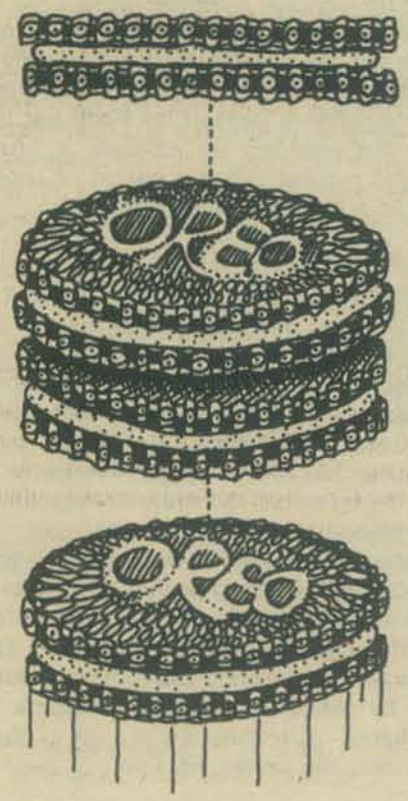
Drs. Dobbs Revisited

Well, dear readers, I went back to Pollo Del Mar and the house of the many Drs. Dobbs. Don't look at me that way! It's my job. And now for the second interview with Drs. Dobb. (The Drs. Dobb spoke alternately.)

Us: I didn't want to come back, but the reports of your new technology, your new computer, made my editor send me here.
Dr. Dobb: Did we do something to offend you last time?
Us: Not again! I'm going to ask the questions!
Dr. Dobb: Would you like a glass of orange juice?
Us: Sure, thank you.
Dr. Dobb: Ha!
Us: ?
Dr. Dobb: I asked a question and you answered it!
Us: Do I get the orange juice?
Dr. Dobb (pouring): Sure. Would you like some cookies?
Us: Try me.
Dr. Dobb: It all started with Dr. Dobb there pulling an Oreo cookie apart.
Us: Huh?
Dr. Dobb: I realized that the creme filling sticks to one side *or* the other.
Dr. Dobb: Usually. Thus we have our OReo gate.
Us: What kind of cycle time does it have?
Dr. Dobb: We observed that a package of oreos has a lifetime measured in microseconds.....
Us: I have made the same observation!
Dr. Dobb: So we soon extended our discovery to a whole family of CMOS devices.
Us: CMOS?
Dr. Dobb: This interviewer sure knows how to play up to a good line.
Dr. Dobb: Sure does.
Us: CMOS?
Drs. Dobb (together): Chocolate Multilayer Oreo Sandwich.
Us: Aha!
Dr. Dobb: And for speed we use OCL.
Us: You mean ECL?
Dr. Dobb: No, OCL – Oreo Coupled Logic.
Dr. Dobb: It is easy to implement stacks.
Dr. Dobb: True. We have been able to get eight oreos on a chip.
Us: How?
Dr. Dobb: Very carefully. Usually the chip breaks after only two or three oreos are on it.
Us: But OR gates are not enough for a computer?
Dr. Dobb: Of course, we have ANDeo's and XOReo's.
Dr. Dobb: How many OR's can you get on a chip?
Us: I've seen a quad OR gate.
Dr. Dobb: If it has only four oars, it's not much of a ship.
Dr. Dobb: I guess if you ship ores you must be in the mining industry.
Dr. Dobb: And you are a miner if you chip ores.
Us: That's terrible!
Dr. Dobb: Did we tell you how we implement a DMA?
Us: Direct Memory Access?
Dr. Dobb: How did you ever guess?
Us: I don't know, but I thought it would be something like Dire Machinations Again. OK, I'll bite, how do you do a DMA?
Dr. Dobb: How is it usually done?
Us: By stealing cycles.
Dr. Dobb: Right! Except we do with stealing cookies.
Dr. Dobb: Hiram (one of the Dr. Dobbs is Hiram Carmichael Dobb) did you hear him say that he'd bite? Why didn't you follow up that possible line for jokes.
Dr. Dobb: Hilary (the other Dr. Dobb is Hilary Cynthia Dobb, and if you'd just remember what you read last issue, you won't need to be told these things), I think jokes on Byte, Nibble and the rest are much too common.

Since this threatened to become a family squabble, I snuck out the back door. Maybe to return again. And maybe not.

Thanx to Kent Strother and Doug Wyatt. We talk like this all the time.



by Jefdragon Raskin

FREE SOFTWARE

OR

by Tom Pittman

SUPPORT YOUR LOCAL SOFTWARE VENDOR

I will be honest. What you are about to read is a gripe, directed primarily at hardware manufacturers. I am in the software business (ITTY BITTY COMPUTERS), so you know my own bias.

Among personal computers there is a software vacuum. Tiny Basic was an attempt to start filling that vacuum. About a year ago PCC started publishing articles entitled 'Build Your Own Basic', giving details and general architecture for Tiny Basic, a children's language conceived by Bob Albrecht. The ideas was to get hundreds of hobbyists across the country working and sharing and producing tons of free software. Instead of hundreds, we actually saw only about a half a dozen successful working programs. Why?

My Tiny is one (or two, depending on your point of view) of that half dozen. I wrote it specifically in response to this software vacuum. It was an experiment, to see if there was any substance to the claim that if software were cheap enough it would not be ripped off like the Altair BASIC. The claim was largely valid; I have reason to believe the ripoff rate is less than 10%. But after nine months of business, I am still not selling anything bigger or better than Tiny Basic. Why?

A quick scan of a couple of hobby magazines shows typically 40 ads offering computer hardware and no software, 10 ads offering both hardware and software (in half of those the software went with the hardware) and only five ads offering software and no hardware (mostly books). Why?

Let me tell you why. Three factors work together to cause these phenomena.

One is that the personal computer market has not matured enough to recognize the importance of software, and there is not the demand for it. Obviously the readers of *DDJ* differ from the rest in this respect (*DDJ* is primarily a software journal), so I will not belabor this point. But even here the emphasis is often on bigger and better systems software instead of considering the system as a tool to achieve other ends.

Second, and more important, is the fact that good software is expensive. It is incredibly complicated, laborious to develop, and impossible to perfect. It is a known fact that the average programmer output is one line of debugged and documented code per hour, so an 8K 8080 program written in assembly language represents two man-years of labor, if done right. Obviously my performance has to exceed that average or I could not stay in business, but the order of magnitude is correct. Somebody has to pay for that cost. The hobbyist software exchange so widely acclaimed in the press (including *DDJ!*) simply has not materialized because the hobbyist is either unwilling or unable to expend the effort and discipline himself to the extent required to produce a program usable beyond his own computer. By comparison, hardware design is a piece of cake (I know – I've done quite a bit of that too). Software is not, and cannot be, free.

The third reason is the object of my gripe. If we want low-cost (to the user) software in abundance, we must find another way to pay for it.

One time-honored way is for the manufacturer to pay for it and bury the cost in his hardware. He offers the software 'free' and hopes (or requires) that the customer will buy enough of his slightly higher-priced hardware to recover his investment. As I mentioned earlier, there were some five or six manufacturers doing this. A more subtle example is that of Processor Technology's 'free' Software Number One: an assembler that ate vast amounts of memory, and guess who sells memory.

Another approach is to produce the software on venture capital, and hope to sell enough of the packages to recover the investment. For what we would consider a reasonable price, a very large volume is necessary. More on this later.

The third alternative is no software. Forty or more purveyors of hardware consider this the way to go. Maybe they hope that 'free software exchange' will happen. I will analyze some of this reasoning a little later.

Now I will admit that the picture is not all that sharply defined in black and white. For example, MITS charges a stiff price for their software AND requires you to buy their hardware. I call them 'the IBM of the Hobby computer field.' Many of the available program products did not cost the developers as much as they should have, and when you try to use the programs you see why. But I said this was primarily a gripe against hardware manufacturers, so let me analyze exactly what is wrong with each of these three software distribution methods.

The best by far from the user's point of view is where the manufacturer provides software with the hardware, either 'free' or as an extra cost option. The hazards are all from the manufacturer's point of view, which probably explains why there are so few doing this. A reasonable cost for 8K BASIC is $25,000. If the manufacturer expects to sell 5,000 computers, he must add $5 to the selling price of each one (that gets marked up to $8 out of the customer's pocket) to recoup his investment. No profits, just break even. All other things being equal, his computer now sells for $8 more than that of his competitor, who offers no software. In a market competing for beer money, price is of prime importance. So the customer accepts his free Basic and buys the competitor's computer to run it in. Now actually, the BASIC probably only cost $15,000 because it was written by a moonlighter, or $5,000 because it was written by a college student as a term project. The other $10,000 or $20,000 will be spent in maintaining the program because the moonlighter got a divorce and went to Venezuela, or the student vanished leaving no documentation. The economics of the situation prevent very much software from being offered by the manufacturer unless the processor is so unique that the higher price will not be noticed and the software will not be lifted to other machines. One of these that comes to mind is the 16-bit machine offered by Dick Wilcox, who has amassed a tremendous amount of software for it by lifting from previous projects.

Other than these sparkling exceptions, the user should expect to get software from other sources as well, which brings us to the third category – the computers with no software (the competitors that the benevolent manufacturers in the first category have to worry about). Many of the buyers of these unpadded (no soft) machines have no use for software. Their only interest is in constructing bigger and (in some mysterious way) better elephants to sit in the attic. I won't worry about them because they represent a relatively dwindling minority of the computer buyers. Of the rest we have a similarly dwindling (though still considerably larger) proportion of hackers who like to build software elephants to metaphorically sit in the attic. They also do not need much supplied software.

The rest of us consider software as tools to be used in the making or doing of other things, be it playing games or writing other programs. Most of us would rather not write a program from scratch if it already exists. This is the market for that second category.

What about the independent software vendor? Look at the economics. Suppose he writes that 8K BASIC. That means investing $25,000 on speculation. Or for someone like myself without a lot of overhead the cash investment is smaller, but we pass up an actual $25,000 job to work on the BASIC instead. In either case the value of the investment is the same. If I can sell it as a package to a manufacturer, that is great: take the money and run. But they are not buying, or at least not for $25,000 up front. They want to offer you 'Royalties'. On what? Not on the increased value your software gives their computer, but rather on the $5 that they will charge the customer for the package. I get letters with some regularity offering me 15% royalties. Fifteen percent of $5 is $.75/unit; since it is priced separately, at least half of the customers will opt not to buy it, so they have to sell 80,000 computers before I show a profit – if I live so long. Perhaps that is a little exaggerated, but I did have a manufacturer of a $1,000 computer tell me that $5 each was too much to pay me for a customized version of my software.

Along this line I should point out that the software

distribution systems that KILOBAUD and others are trying to set up are unlikely to attract much Grade A software for very much the same economic reasons. But there are other problems.

Well, then, the software vendor must go directly to the public. A recent census has it that there are 15,000 computers out there. If I sell this BASIC at $10 a copy, I might get $5 each, so I need to sell 5,000 copies. Considering that of those 15,000 computers probably less than half have 8K of memory, and of those with the requisite hardware, less than half will buy the program (actual experience seems to run closer to 10%), there is already no way I can sell enough copies to recoup my investment. OK, so we call it a 5-year product life, and estimate 100,000 computers by 1980.

Let us assume further that many of those 100,000 computers still come with no software (though the trend is away from that), and because this is actually worth its $25,000 (as opposed to $5,000 or $15,000), it is recognized as one of the better products on the marketplace and captures 40% of the market from the other six or eight BASIC interpreters out there, some of which are 'free'. There still remains the fact that there are at least three different CPU chips going into those computers.

Finally, the Gotcha: Every one of those manufacturers with less than cradle-to-grave software support (i.e., most of them) designs the hardware configuration just a little bit differently, so that none of the programs that work on one computer will work properly and conveniently on the next, EVEN THOUGH THEY USE THE SAME CPU CHIP! I can understand a manufacturer designing in a new bus structure to lock the customers into buying high-profit peripherals from the same source as the low-profit CPU. But considering the costs of software development versus the potential profit, I should think the manufacturers would try to make portability as easy as possible, so that their computer could be used with someone else's software. An obvious exception here is MITS: their BASIC is generally accepted as one of the better packages, and it is also quite expensive. A little software incompatability (and they did it, too!) might conceivably reduce the chances of BASIC walking to other computers.

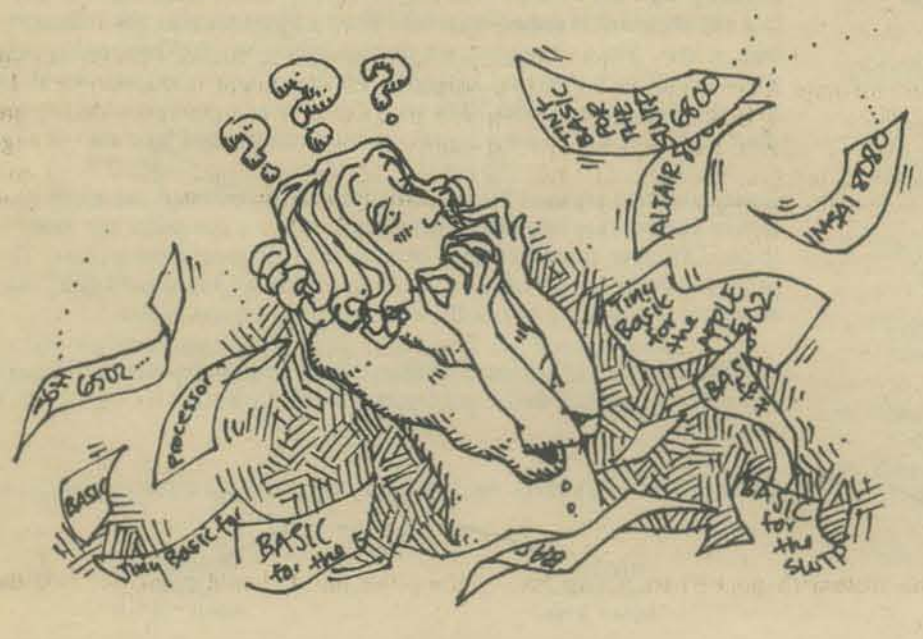
So now the software vendor has to support different versions of his package for every different manufacturer. I have seven different versions of Tiny Basic, and there are still at least four more 6800 and 6502 systems that I do not directly support – all different. I have no idea how many different varieties of 8080 and Z-80 systems there are, but I'll bet there are at least six.

What can be done about this mess? First, you manufacturers can adhere to a few simple conventions:

1. Put your monitor in high memory, and leave low memory in RAM.
2. Avoid using memory Page 00 for any system software or firmware, but if you must use some restrict it to the first and/or last 20 or so bytes.
3. If the monitor needs additional RAM for local storage or buffers, put it at the far end with the monitor ROM.
4. Start the stack pointer at the highest available memory address, so it has the most freedom to grow without interfering with low memory.
5. Provide monitor subroutines to:
   a. Read a character from the console keyboard.
   b. Display a character on the console output device.
   c. Terminate the current line of display and start a new one.
6. Always use the A register (accumulator) to pass one-byte data to and from monitor routines. Avoid altering the other registers.
7. Use a Jump vector at one end of your monitor to define the entry points to monitor subroutines, so that subsequent revisions will not affect previously developed software packages.
8. Provide at least one standard program loading peripheral such as paper tape reader or Kansas City audio cassette, even if you do not include a way to generate output media in the same format. There is more need to load package software than there is to produce it. Your Super High Speed Patented Data Recording Technique is fine for local data and program storage.
9. Support the CPU chip manufacturer's conventional hexadecimal program format for that program loader hardware. Again, save the tricky double-checksum reverse binary format for local program and data storage. The software vendor cannot afford to support 23 different media standards, but there is nothing wrong with requiring each buyer of a package program to read it in slowly, then recopy it to his fast local storage peripheral.

Those of you who are not manufacturers, you can help too. Complain at nonconforming manufacturers. Refuse to buy their products (remember, you have to interface package software to the non-conforming systems, and eventually the software vendors will only support good-guy systems).

If we in the software business can ever manage to concentrate on new products that have a reasonable chance at high volume, you might begin to see more and better low-cost software. And when that happens more people will be willing to buy personal computers, which will make the software market more attractive, which in turn will result in more and better software. It's up to you now. ■



BAY AREA COMPUTER EDUCATORS (BACE)

by LeRoy Finkel

Three years ago, a small group of San Francisco Bay Area computer education leaders began a regular series of meetings to share ideas, concerns and information. Representation included area high schools, community colleges and the University of San Francisco. During year one our effort was devoted to sharing programs and knowledge with the end result of everyone contributing their good programs into a community 'pot' which was made available to all members.

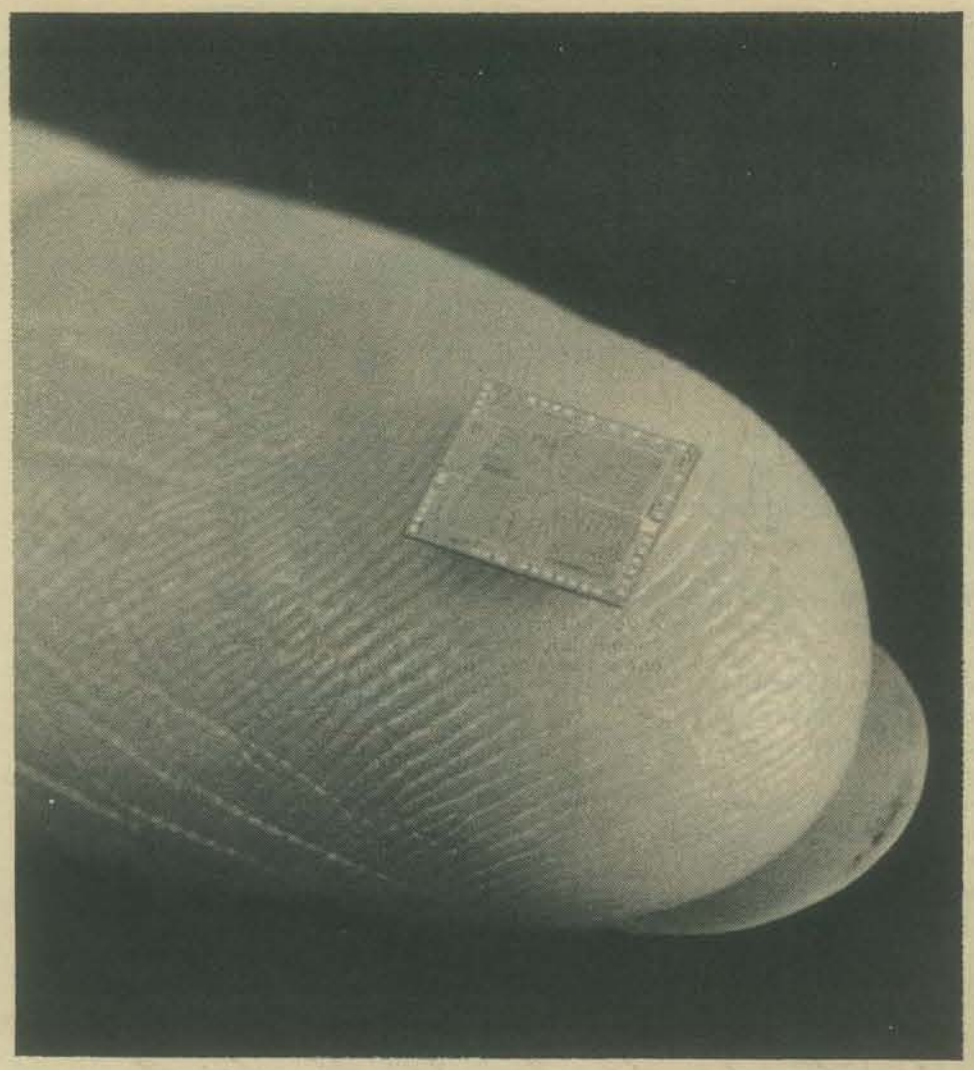
During year two we focused all of our effort on the evaluation of computer guidance systems which was, at the time, the 'hot button' for all of us. The result of this effort was the creation of EUREKA, a California version of the Oregon Career Information System (C.I.S.). If all goes per present schedule EUREKA will be available to California users in Spring, 1977. It should be noted that the Californication of Oregon's programs has been done without the benefit of state or federal funds. It has been a bootstrap effort, successful because of the hard work of a few dedicated people. Grant monies are being sought to aid the dissemenation and updating of EUREKA.

This is year three. Our goal is to explore the uses of computers in Special Education (physically handicapped, educationally handicapped and mentally retarded). Our expected outcome is to catalogue *what* is available for use in this area and evaluate same. We may even develop our own programs if time/need permit.

If any of you are using computers with identifiable Special Education students, please write us with information about your efforts. We will gladly share our findings with you.

Our address:

LeRoy Finkel
BACE
c/o P.O. Box 310
Menlo Park, CA 94025



CHIP TALK REVISITED

BY DON INMAN

Microprocessor - What is it? According to Sippl and Sippl [1], a microprocessor is "a device capable of receiving data, manipulating it, supplying results usually of an internally stored program." In the words of Brice Ward [2], "A microprocessor is a very small processor, and a processor is a special machine that has been devised for the express purpose of processing information – of performing specific tasks. It represents an extremely powerful and inexpensive design approach to a wide variety of industrial, commercial, recreational and educational applications."

The microprocessor is often referred to as a *microcomputer*. Although it does contain the arithmetic and logic functions associated with computers, it is not really a microcomputer. In the typical microcomputer, the processor is the heart of the computer. It is contained in a single chip (integrated circuit - sometimes referred to as an IC).

Such chips are called *microprocessor chips* or simply *microprocessors*. They are used in the construction of a microcomputer, minicomputer, or even peripheral devices or full size computers. To a large extent the microprocessor determines the characteristics of the microcomputer of which it is a part.

"The modern microprocessor is the product of two technological developments: Large-scale integration (LSI) and low cost semiconductor memories." LSI and semiconductor memories are described in the last issue of PCC. [3] Through these developments it has been possible to design processors which will routinely handle complex problems. The manufacturing techniques have made these chips relatively inexpensive while retaining sophisticated capabilities. Inexpensive semiconductor memories have provided a compact way for microprocessors to store instructions and data.

As a result of these developments, microprocessor circuits have been designed with relatively few integrated circuits. When memory and input/output (I/O) devices are added, it is possible to put the entire microcomputer on a single printed circuit board.

Microcomputer - What is it? Everyone agrees that it is a computer, but there the general agreement ends. The terms "microprocessor and microcomputer" have been so loosely used that they have become almost interchangeable. The microcomputer is something more than a microprocessor.

One could say that a microcomputer is merely a very small computer, but they are really a new and different product. The use of logic chips and the price of microcomputer devices becomes the most important distinction. They do share common ancestry however.

Another "gray" area exists in the difference between microcomputers and calculators. Motorola Semiconductor [4] makes this distinction: "Any hardware system will be designated computer as opposed to calculator when the following conditions are fulfilled:

- it has a random access memory for read/write operations;
- it has a controllable input-output system;
- its repertory of instructions allows:
  a) the manipulation of words stored in the memory (arithmetic, logic or transfer operations)
  b) the modification of any bit in a word;
  c) transferring the control of a programme by branching when the necessity arises (decision making power of a computer)
  d) controlling the external equipment with the aid of an interruption facility;

the instructions, that is the programme, are stored and processed using the same hardware as for the data.

This definition already allows us to get a first idea of the structure of a computer or of a minicomputer, which has the same organization as a big computer but differs from it essentially as regards price, performance and the field of applications which it covers."

With the variety of definitions for a microcomputer, it becomes necessary to examine what a manufacturer is including when a microcomputer is discussed. Is it just the microprocessor, is it the microprocessor and closely associated chips, or is it a complete microcomputer?

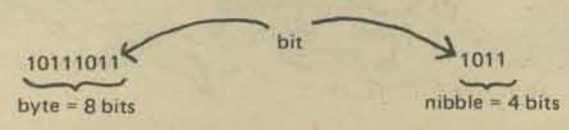
Bits, Bytes, Nibbles & Words - "The reason there is no fundamental difference between a microcomputer and any other computer is because all computer products are based on the same fundamental computing concepts – which in turn devolve to one fundamental logic concept – that of the binary digit." [5]

A binary digit can be expressed as either a 0 or a 1. Binary digits can be likened to a switch which is either on or off. Binary digits are also like voltages - either high or low. Since computers are electronic devices, they can readily take advantage of such electrical analogies. Numbers larger than one can be expressed by a string of zeros or ones. The term "bits" can be considered as merely a short nickname for "binary digits". A bit is represented by a zero or a one.

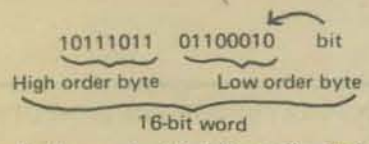
Binary numbers are used by the computer. These numbers are made up of bits. Binary numbers are organized into groups so that a computer can easily handle them. Different computers may operate with different sized groups. The particular group used by a given computer is known as its word size. An 8-bit computer is organized to handle words which are 8 bits in size.

An 8-bit data unit is commonly referred to as a *byte*. In other words, as the computer is handling data it processes the data by bytes. It "bytes" off 8 bits at a time.

Similarly 4-bit data units, used by 4-bit computers are often referred to as *nibbles*.

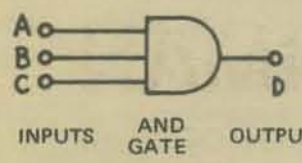


If a microcomputer has an 8-bit wordsize, the terms "byte" and "word" can be used interchangeably. If a computer has a different word size (i.e., not eight bits), then "word" and "byte" do not mean the same. A *byte* refers to an 8-bit unit. A *word* refers to the unit of a computer's word size. A "16-bit" microcomputer uses a 16-bit word made up of two 8-bit bytes.



These little gadgets are electrical "gates" which can be opened or closed to control circuits. By using them in microcomputer circuits, we can control the communication of bits, nibbles, bytes, and words.

There are several kinds of gates. One is the AND gate. It will "pass" or, output, a high signal (1) only if all its input lines have a high (1) signal. The figure below shows a three input AND gate and its truth tables. Truth tables are commonly used to show how gates work. Each line shows a condition for the inputs and output.

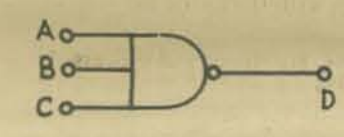


Truth Table

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

As can be seen from the table, the output will always be low (0) unless all the inputs are high (1). AND gates may have more or less than the three inputs shown. In any case, the output will only be high (1) when all the inputs are high.

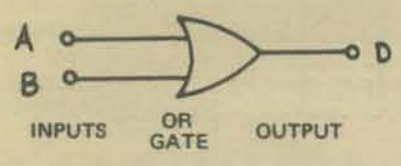
Another kind of gate is the NAND gate. The output of the NAND gate is the complement (opposite) of the AND gate. By complement we mean: 1 is the complement of 0. 0 is the complement of 1. The symbol for the NAND gate is similar to the AND. The small circle at the output indicates the complement.



Truth Table

A	B	C	D
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

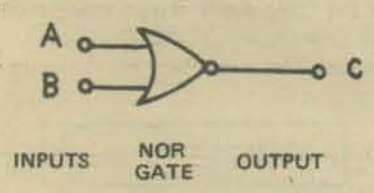
The OR gate produces a high (1) when one *OR* more of its inputs are high (1). OR gates may also have several inputs. A dual input OR gate is shown.



Truth Table

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

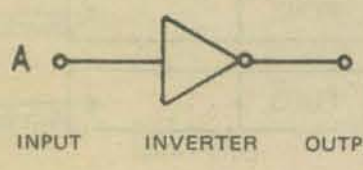
The NOR gate produces the complement of the OR. It produces a high (1) output only if all inputs are low (neither A *NOR* B).



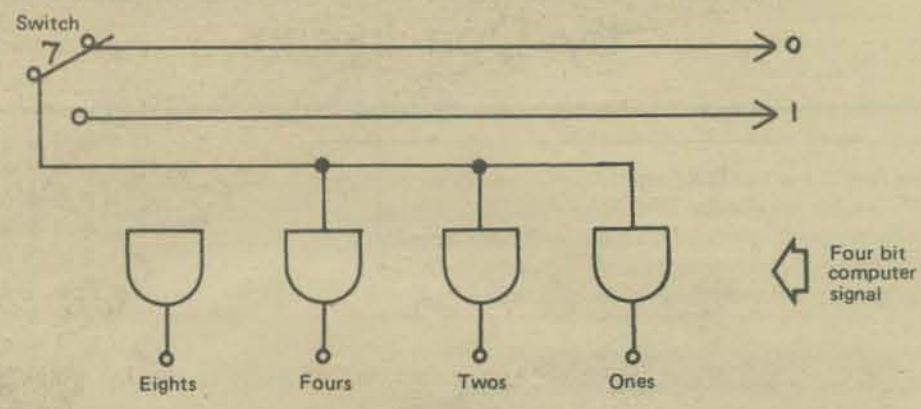
Truth Table

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

The last gate we will discuss is the INVERTER. It has only one input and one output. If the input is 1, the output is 0. If the input is 0, the output is 1. Its symbol is:



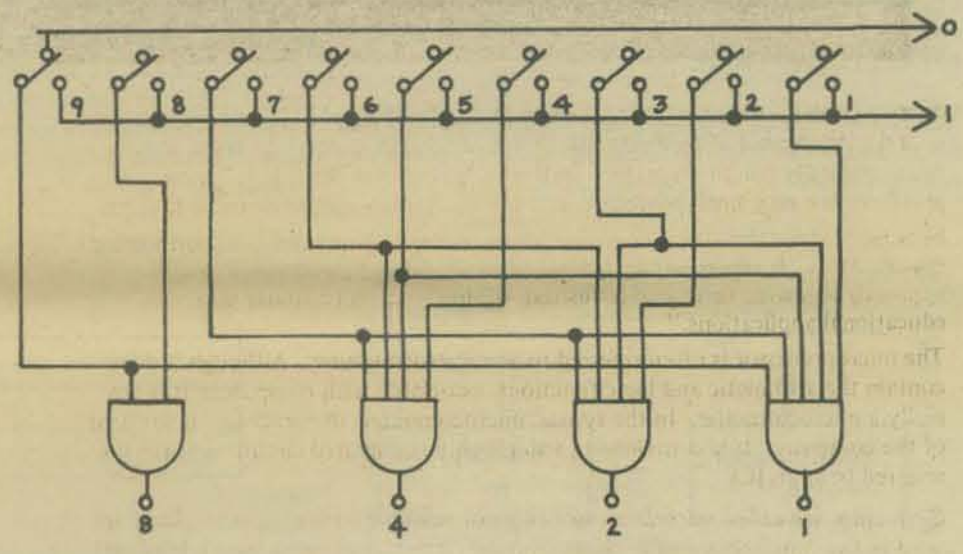
A typical example of how an AND gate might be used would be for data entry conversion to binary numbers. The keys on the entry board might be labeled 1 through 9. Single key connections might be as shown for the number 7.



A dot indicates a physical connection of wires. The one's, two's and four's bits *only* are connected for the 7 switch.

When the 7 key is depressed, the high (1) signal is connected to the appropriate AND gates, and the computer sees the binary number 0111.

Key connections for the decimal numbers 1 through 9 could be connected in the following fashion.



Each switch is connected to the appropriate gate to give the correct binary code when pressed.

REFERENCES

1. Sippl, Charles J. and Sippl, Charles P. *Computer Dictionary and Handbook.* Indianapolis: Howard W. Sams & Co., Inc.–The Bobbs Merrill Co., Inc., 1972.

2. Ward, Brice. *Microprocessor/Microprogramming Hand Book.* Palo Alto, California: Tab Books, 1975.

3. "Chip Talk," *People's Computer Company,* Vol. 5, No. 3 (Nov.-Dec., 1976). Published by People's Computer Company, Box E, Menlo Park, CA 94025.

4. "Motorola Semiconductor Manual," *SCCS Interface,* January 1976.

5. Osborne, Adam. *An Introduction to Microcomputers, Volume I.* Berkeley, California: Adam Osborne & Assoc., 1976.

6. Libes, Sol. *Digital Logic Circuits.* New York: Hayden Book Co., 1975.

THE DATA HANDLER USER'S MANUAL: PART 2
by Don Inman

Don Inman is a teacher on sabbatical who's been working with teachers in the San Jose School District. Under Don's guidance, the teachers have built Data Handlers, complete microcomputer systems based on the 6502 microprocessor, and are now learning to use them.

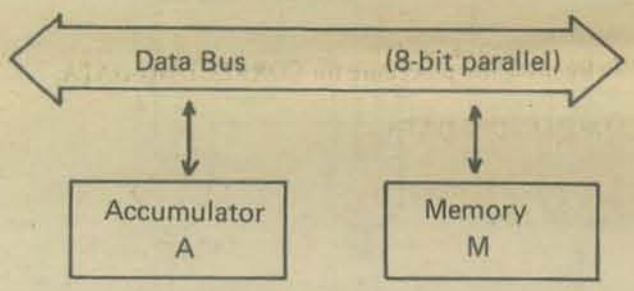
This user's manual is designed to serve both as a self-teaching guide and as an outline for a course at the beginning level of computer science. While it deals specifically with the Data Handler, it can easily be adapted to other microcomputers using the MOS Technology 6502.

The first semester course consists of nine two-hour class sessions, the first two of which were spent constructing the systems. Volume I of the Data Handler User's Manual covers sessions three through nine; Volume II will cover a second semester course. The third session of the course was covered in the last PCC; here's session IV.

SESSION IV – TRANSFER OF DATA

WITHIN THE MACHINE

The data bus is used to transfer data between memory and the processor's internal registers, such as the accumulator, as shown below.



All operations between memory locations must be communicated through the accumulator or one of the auxiliary index registers (x and y). Data can be brought into the accumulator, operations can be performed on the data, results can be tested, new bits set into it, or it can be tested, new bits set into it, or it can be transferred back out of the accumulator. The accumulator serves as an interim storage for a series of operations such as adding two numbers (one is loaded in, the second is added, and the result is obtained and temporarily stored).

Two instructions which we will be using to demonstrate the transfer of data by means of the accumulator are:

1. LDA (Load Accumulator)
2. STA (Store the contents of the Accumulator)

LDA and STA are mnemonic codes used for the 6502 instruction set. These codes save time and space when writing programs.

INPUT TO THE MACHINE

The hexadecimal keyboard is an integral part of the Data Handler. The sixteen keys in the lower right corner of the board are used to input data to our computer. The switching circuits are wired so that each key produces a binary bit pattern unique to that key. Four bits are used to produce each hex symbol. These patterns are as follows:

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

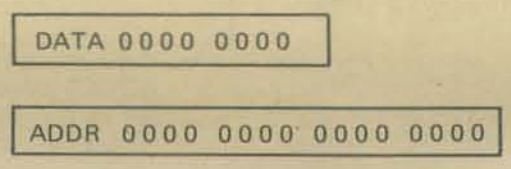
The combination of two 4-bit hex digits produces the 8-bit byte of data to be transferred. Keep in mind that data is loaded, transferred, and stored in 8-bit bytes. These bytes are transferred via the data bus in a parallel fashion (all eight bits together rather than one bit at a time.)

Other devices can be used to INPUT data by means of the eight-bit input port in the upper right corner of the Data Handler board. We will discuss the use of this port in Volume II of the User's Manual. For the time being our goal is to familiarize ourselves with the Data Handler itself.

OUTPUT FROM THE MACHINE

Both INPUT and OUTPUT can be monitored by means of the DATA and ADDRESS L.E.D.s located on the left central portion of the Printed Circuit Board. INPUT and OUTPUT are accomplished by means of the computer's memory. Therefore, to see the output of the computer, we must examine the specific location in memory which contains our result.

The data and address L.E.D.s are configured on the printed circuit board as shown:



The data L.E.D.s (light emitting diodes) are the eight in the top row. They are arranged in groups of four to represent a hex digit. The lights will show the data which is residing in the current address. The address and data information are used together. The address lights show which location of memory you are looking at, and the data lights show the contents of that address location. In both cases, the most significant bit (highest place value) is located on the left. The least significant bit is located at the right.

The Data Handler has 1K of random access memory (RAM) on the board for user's programs. It begins at the hex address FC00 and ends at FFFF. The locations FFFA through FFFF are reserved locations. DO NOT store programs in that area. Locations 7F00 through 7FFF are decoded for INPUT/OUTPUT devices. The Data Handler uses 7FFE for its output port and 7FFF for its input port. User devices may be connected to these ports in the upper right corner of the board. Output devices are discussed in the last section of this manual. Input devices will be covered in Volume II.

It is probably a good idea to start your programs in the first available location FC00. Using this convention, you will have the largest amount of available memory to work in. Lower numbered memory locations are not used in the Data Handler unless the user adds his own memory boards. This can be accomplished through the 100 pin connector option at the left rear portion of the board. Altair/IMSAI memory boards can be added at that point.

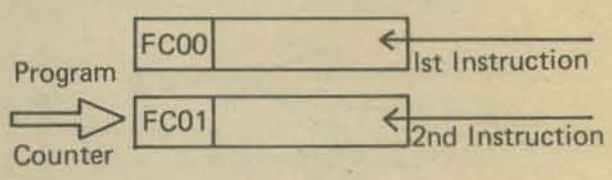
INTERNAL OPERATION OF THE COMPUTER

Each instruction is made up of an 8-bit binary word or number that is recognized by the control unit of the computer. The control unit responds to the instruction by sending out the appropriate control and timing signals which cause other sections of the computer to move and manipulate data. The programmer must specify to the computer exactly what he wishes the computer to do. This is done by writing a sequence of binary instructions called a program. By means of a carefully prepared program the computer can perform complex tasks.

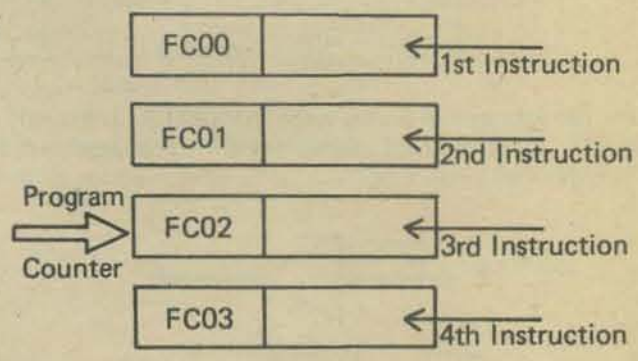
A machine language program is written for the Data Handler in hexadecimal format. Through the keyboard entry, the instructions are translated into binary number codes and stored in the computer's memory. The memory can be thought of as a set of storage boxes arranged in order and numbered with the hex values FC00 through FFFF. Although the Data Handler has 1K of memory located at these values, you may have a computer with memory at other locations. Using the Data Handler, we begin with the box numbered FC00. Each memory box will contain one instruction or byte of data from the program. See the diagram 'Program Input' at the top of the next page.

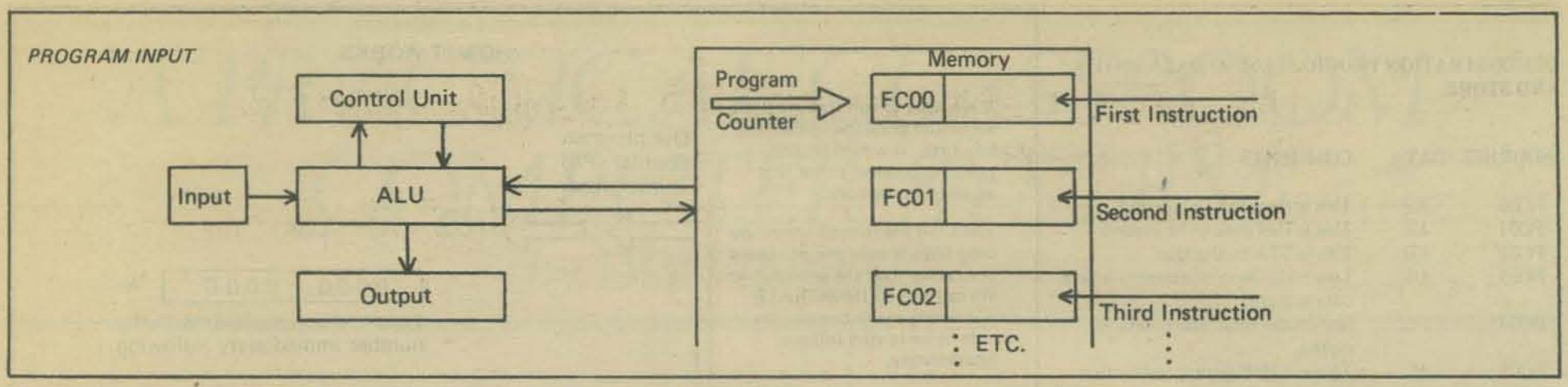
The control unit contains a binary counter (called the program counter) which is set to the memory location of the first instruction by the computer's initializing process (see LOADING PROGRAMS, steps 9-15). The memory is then searched to find this particular location. The contents of this location are then sent to the control unit. The control unit decodes the instruction and sends out timing and control signals. The computer responds to these signals in a predetermined manner to implement the instruction. After this instruction has been executed, the program counter is incremented to the next sequential location.

After executing the instruction at FC00, the program counter is incremented by one and now 'points at' FC01.



The contents of this location are now sent to the control unit, and this second instruction is executed in the same fashion as the first. The program counter is incremented by one and now 'points at' FC02.





The computer proceeds in this fashion, incrementing the program counter after each cycle until the program has been executed in its entirety. That part of the process which sends out the contents of the program counter and retrieves the instruction from memory is called the FETCH CYCLE. That part which actually executes the instruction is called the EXECUTION CYCLE. The fetch cycle is automatically performed in the same manner for each instruction. The computer's action during the execution cycle is different for each instruction (i.e., the computer responds differently for an ADD instruction than it does for a SUBTRACT instruction.)

LOADING PROGRAMS

The procedure for loading programs is given below along with a block diagram indicating the action which takes place for each keyboard operation. The block diagram is an over-simplification of the actual operation which takes place.

Keyboard procedure:

1. Press the CL key (This clears the internal mode and data holding registers.)
2. Press the AD key (This switches loading to the address lines. If the address mode light is already on, this step is unnecessary.)
3. Key in your starting address for the program (Usually FC00)
4. Press the DA key (This switches loading to the data lines. The data which is to be loaded will be put in the address selected in step 3.)
5. Key in the first byte of data (Two keystrokes.)
6. Press the DP key (This deposits the data on the data lines into the address on the address line.)

The block diagram attempts to show the order of the computer's operation as the various keystrokes are made.

Our first byte of data has now been loaded into the first memory location. At step 6 when the data was deposited, the address selector was incremented by one (this accomplished the same thing that steps 2 and 3 did before). Therefore, since we are still in the data loading mode (from step 4), all we have to do to deposit the next number is:

7. Key in our second byte of data
8. Press the deposit key

Step seven puts data on the data lines, and step eight deposits that data into memory and steps the address to the next location.

Steps 7 and 8 are then repeated until the entire program has been entered.

The only remaining data to be entered is our initializing vectors which tell the computer where to start (what memory location) our program. So we then:

9. Press the AD key
10. Key in FFFC (this is where our first location will be found.)
11. Press the DA key.
12. Key in the starting address (low order byte) 00
13. Press the DP key
14. Key in the starting address (high order byte) FC
15. Press the DP key

This completes the program loading. If any data is to be accessed from outside the program, it would be loaded at this time. If no mistakes had been made, we would now be ready to RUN the program. However, the chance of loading a long program perfectly is rather low. We should always check programs by switching to the examine mode.

EXAMINING PROGRAMS

After your program has been entered, it should be examined for possible errors with the following procedure:

1. Press SC or SI key. (The computer must go through at least one cycle before we can use the examine mode.)
2. Press EX key. (Puts computer in EXAMINE mode.)
3. Press AD key. (To place keyboard entry on address bus.)
4. Key in the address to be examined. (Normally the first location of the program.) THE DATA IN THAT PARTICULAR ADDRESS WILL APPEAR IN THE DATA LIGHTS. THE ADDRESS WILL APPEAR IN THE ADDRESS LIGHTS.
5. Press EX key. (This increments the address by one.) THE DATA IN THIS NEW ADDRESS APPEARS IN THE DATA LIGHTS. THE ADDRESS APPEARS IN THE ADDRESS LIGHTS.

Step 5 is repeated to examine each successive address (memory locations). If you want to examine an address that is *not* in sequential order you must:

6. Press the AD key again.
7. Key in the address to be examined.

After examining the entire program, errors can be corrected by using the procedure for CORRECTING DATA.

CORRECTING DATA

1. Press CL key
2. Press AD key
3. Key in the address of the data to be corrected
4. Press DA key
5. Key in the corrected data
6. Press the DP key

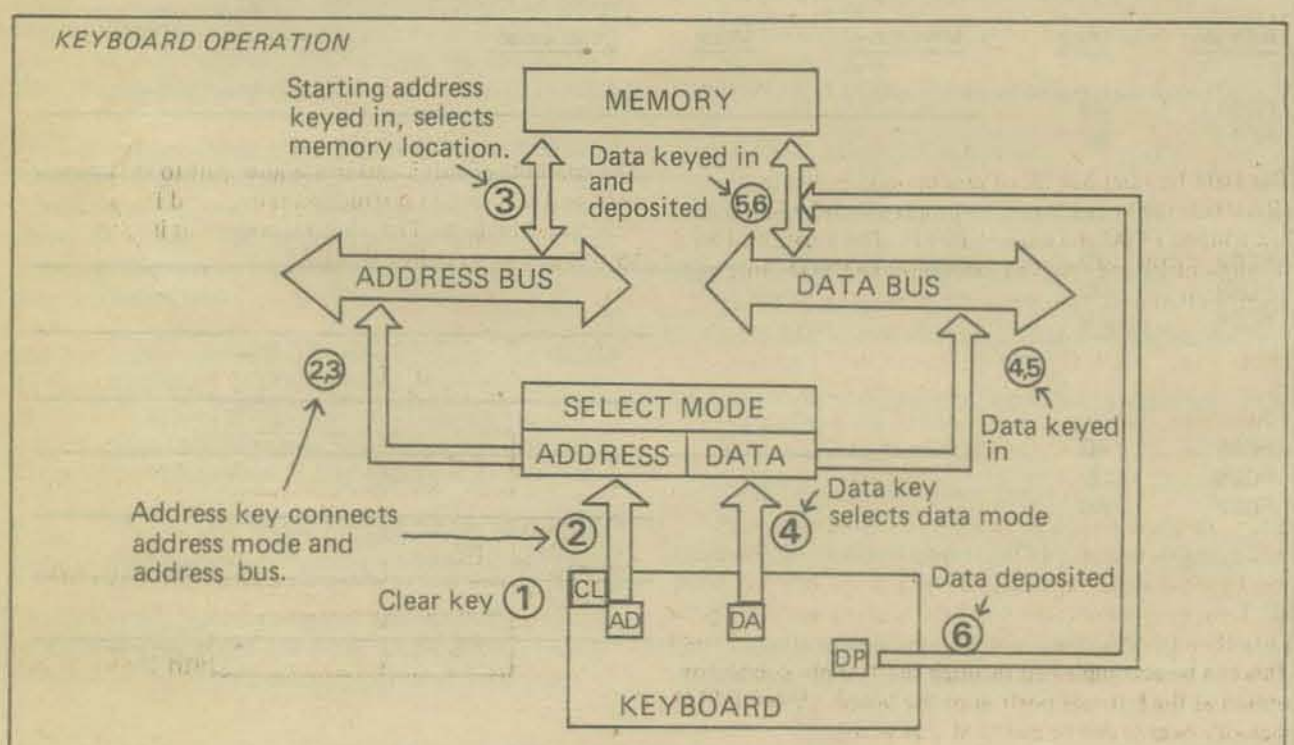
Repeat steps 2 through 6 until all errors have been corrected. At this point it is a good idea to repeat the EXAMINING PROGRAMS procedure to be sure all errors have been corrected. After this has been done, we are ready to RUN the program.

RUNNING PROGRAMS

This is the shortest and easiest procedure we have discussed so far. All you have to do is:

1. Press INT key. (This initializes the computer, i.e., locates the correct starting address.)
2. Press the RN key (the yellow RUN MODE light goes on.) After a few seconds
3. Press HT key (the yellow light should go out. This is the Halt Key. If the yellow light doesn't go out, fiddle with the HT and INT Key. Sometimes if you press them both and then release the INT Key, it will halt.)
4. Use the EXAMINING PROGRAMS procedure to locate your result (assuming you stored the result in some memory location).

You now know how to load, examine, correct, and run programs so let's get started. We will start with a program which merely moves some data within the machine.



DEMONSTRATION PROGRAM USING DATA ENTRY AND STORE

ADDRESS	DATA	COMMENTS
FC00	A9	This is the LDA instruction
FC01	13	This is Hex data to be loaded
FC02	8D	This is STA instruction
FC03	10	Low order byte of memory where data is stored
FC04	FC	High order byte where data is stored.
FC05	4C	This is a JMP (jump) instruction
FC06	00	Low order byte of jump destination
FC07	FC	High order byte of jump destination

The program must now be loaded into successive memory locations as indicated by the addresses given. To do this we clear the internal mode, then switch to the address lines as discussed in the section LOADING PROGRAMS. Follow those directions and load the program. Don't forget steps 9 through 15 which put the correct values in the initializing vectors.

When you have loaded the program, use the EXAMINING PROGRAMS procedure to make sure all your entries are correct. After this has been done, if there are any errors correct them with the CORRECTING DATA procedure. Last of all, use the RUNNING PROGRAMS procedure to make the RUN.

After halting the computer, use the EXAMINING PROGRAMS procedure to check address FC10, which should now contain the hex number 13 which was originally entered.

What did our program really accomplish? We loaded the accumulator with the number 13. Then we stored the accumulator's contents (13) into memory location FC10. We then told the computer to go back and do the same thing over and over again. It will continue to loop like this until the Halt (HT) key is pressed. All the program accomplished was to put a number into the accumulator and then move it into a memory location. We used three instructions of the 6502 instruction set. They were LDA, STA and JMP. These symbols are called MNEMONIC codes because they are abbreviations of the computer actions taken and help us to remember the instructions. They are easier to remember than the HEX numbers which we actually enter from the keyboard.

Let's break down the program into each of the individual instructions to see what each does. A diagram of activities is shown on the right of the page. Follow it as we go through each instruction.

If you followed the step-by-step instructions under 'How It Works' you should be ready to try a program on your own. The 'Homework' program at the right is similar to the one we have been discussing. The steps are arranged in groups. Each group causes a specific action by the computer. In the blanks at the right, give the MNEMONIC CODE of each instruction, its ADDRESS MODE, and describe briefly what each group does. Answers are given on Page 33. Once you're satisfied that your program is correct, try running it and see if the data is transferred.

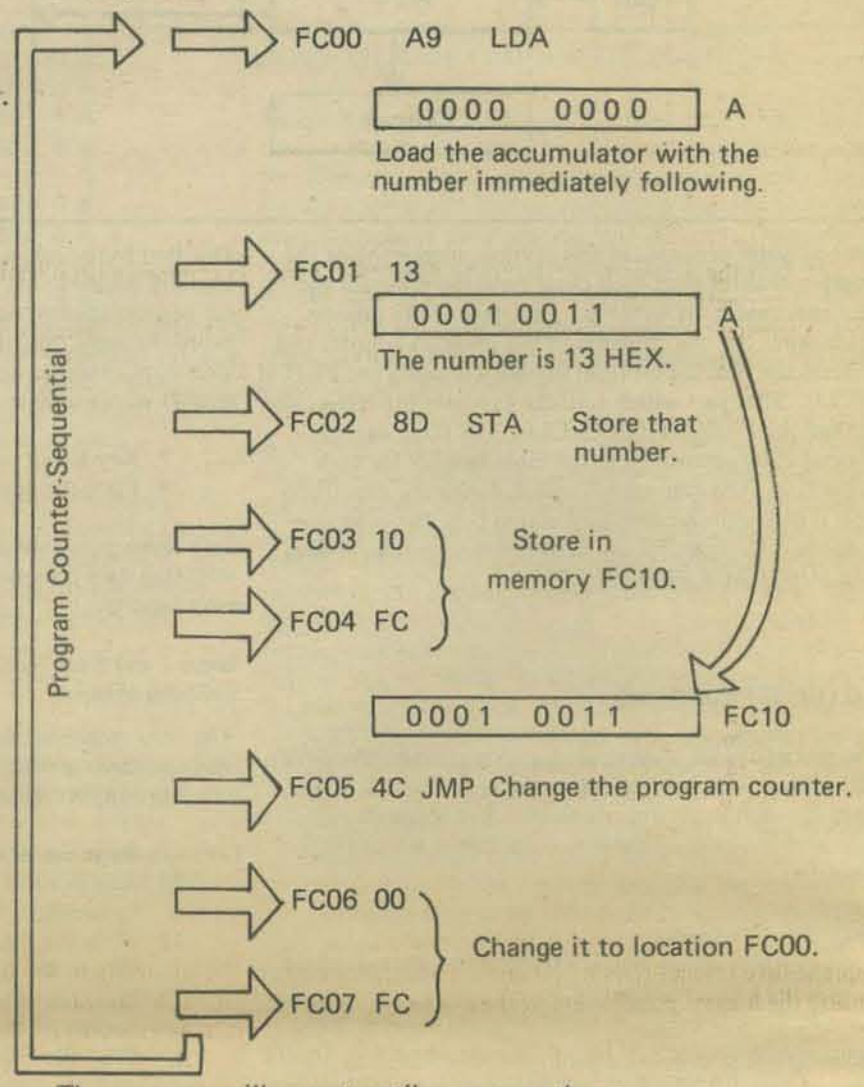
You now know how to load and store numbers. In the next session we will tackle our first practical program. Simple addition will be explained first. There is only a minor revision to the program we have been using above. We will load one number, add a second number to it, and store the result into a specific memory location.

HOW IT WORKS

If we were to run this program, we would press the initializing key first. It would set the program counter at the first location in memory.

Our program counter (PC) is initialized.

The LDA instruction which we used (A9) is only one of several codes that load the accumulator. We call this an IMMEDIATE addressing mode because the data to be loaded follows immediately.



The HEX number 13 is loaded into the accumulator.

The STA instruction 8D is an ABSOLUTE mode instruction which stores the data (13) into memory location named next. The exact, ABSOLUTE memory FC10 is used here. Note the low order byte is entered first. The number 13 is still in the accumulator but has now been COPIED into location FC10.

The JMP instruction is also an ABSOLUTE addressed mode since we told the computer ABSOLUTELY which location we wanted it to go to FC00. The program counter is actually changed to this number forcing the program to repeat itself.

A computer program ordinarily runs sequentially until commanded to change by an instruction such as a jump.

The program will repeat until you press the halt key. (HT)

HOMEWORK

Address	Data	Mnemonic	Mode	Description
FC00	A9	_____		
FC01	36			_____
FC02	8D	_____		
FC03	05			
FC04	FD			_____
				_____
FC05	4C	_____		
FC06	05			
FC07	FC			_____
				_____
				_____

A PRACTICAL CLASSROOM COMPUTER SYSTEM

by DON INMAN

This series of articles will present computer systems which are available at the present time. It will be devoted to systems which are actually in use in the classroom. User reactions (successes and failures) will be discussed in future issues. Computers in several price ranges will be discussed.

One of the prime considerations in the selection of equipment was the ease with which it could be used. Every teacher does not have the required technological background to make unrelated pieces of equipment. Our basic philosophy has included the belief that the simpler the equipment is to operate - the more it will be used. Reliability of the equipment is paramount in the selection process.

The Sunnyvale (California) School District (for K-8 schools) has acquired a typical microcomputer system for use in the classroom. Although purchased with MGM (Mentally Gifted Minor) funds and used exclusively by MGM students, it is a system which is practical for general classroom use.

I have chosen the Sunnyvale system to begin the series since I have been involved in the purchase and construction of the equipment (also see PCC, Vol.5, No. 2). A comprehensive teacher-training program is being prepared to insure the highest possible use of the system.

The Sunnyvale system is priced in the neighborhood of $2500. Two systems were actually purchased. One contains 8K of RAM memory, and the other contains 12K of memory. Otherwise the systems are the same. A brief description follows:

- IMSAI 8080 Computer
- Processor Technology 4K and 8K memory boards
- National Multiplex Serial Input/Output board
- National Multiplex Cassette Tape Recorder
- Lear-Siegler ADM-3 I/O Terminal
- MITS BASIC language on Cassette Tape

BASIC language was chosen as the communicating vehicle because of its simplicity and the volume of BASIC language programs available. MITS BASIC was chosen since it is recognized as being one of the most powerful available.

The Lear-Siegler terminal was chosen because of its commercial acceptance and use, as well as its economy. A video output was desired to prevent undue noise in the classroom setting. This terminal is easy to interface with any computer, having both RS-232 and 20 milliamp current loop communication.

The National Multiplex I/O board was chosen for its ability to control Input and Output to the tape recorder. It has control capabilities through PROM which can be quickly and simply accessed. The National Multiplex Recorder provides compatibility with the I/O interface.

Processor Technology memory boards were selected because of their quality and their reputation for success.

The IMSAI 8080 microcomputer was selected for its combination of quality and low price. The 8080 has been established as the leading microprocessor with the most available software. The IMSAI version has been proven in commercial applications.

To make the system operational in BASIC, only a few steps are necessary:

- Turn on power to Recorder, Terminal and Computer
- Press reset switch on Computer
- Raise two Address switches
- Raise Examine switch
- Raise Run switch
- Type load command on the Terminal
- Start the recorder
- Within a minute the BASIC is loaded
- The Recorder is stopped and the Computer started

That's all there is to it. BASIC is then available and user programs can be typed in.

The recorder provides several routines to control the tape recorder to save programs, load programs, and run programs.

The simplicity of the operating system requires much less time to train teachers in the system's use than might otherwise be necessary. This means that student operation will also be accomplished with a minimum of training. Use of the equipment then becomes the prime and immediate objective.

In addition to the computer equipment, Sunnyvale purchased reference and textbooks to enhance the progress of their program. The computer library now includes the following:

BEGINNING BOOK LIST*

My Computer Likes Me When I Speak In BASIC; Bob Albrecht; 1972

Teach Yourself BASIC, Vol. 1 & 2; Bob Albrecht; 1970

Basic BASIC; James S. Coan; 1970

BASIC Programming; Kemeny and Kurtz; 1971

What To Do After You Hit Return; PCC; 1975

Computer Lib/Dream Machines; Theodore Nelson; 1974

Computers and Computation; Scientific American; 1950 - 1971

101 BASIC Computer Games; David Ahl, Ed.; 1974

The Best Of Creative Computing, Vol. 1; David Ahl, Ed.; 1976

*All available from PCC Book Store, PO Box E, Menlo Park, California, 94025

At the terminal is Dr. Rachel Hamlin, Director of Program Development for the Sunnyvale School District.

The completed system (of which there are two) in the Sunnyvale district

Checking out the IMSAI 8800

A HIGH SCHOOL COMPUTER SYSTEM

by CHRISTOPHER LETT

In late May of 1975, John F. Kennedy High School in Somers NY suddenly had to face a computational crisis. For the previous two years, the school had been given computing time gratis by the local Board of Cooperative Educational Services; our only expenses were the purchase of an acoustic coupler, and the telephone connection costs. But that May we were abruptly informed that the service would no longer be available.

The problem was that there were three courses that made use of the computer already scheduled for the following school year: an interdisciplinary course, a full year calculus course, and a course in BASIC language programming. Since it was too late to design new courses and drop these, the school began to search for an inexpensive computing system. Besides low cost, we needed a system with a powerful, high level conversational language (either BASIC or APL) with the ability to store programs in some form such as paper tape or audio cassette. The terminal had to provide hardcopy and come with a paper tape reader, if necessary.

Examining the Alternatives

The first possibility was purchasing computing time on a time sharing basis from a major corporation. This would have cost the school over $3000 per year, and was therefore rejected as being too expensive.

The second alternative was to purchase a self contained computing system, such as the IBM 5100 or the Wang 2200. Although they would have filled most of our requirements, their high initial costs ($9000 and $5400 respectively) made them again too expensive for our small private high school's tight budget.

The third and most probable choice was to buy a minicomputer with BASIC software and rent a teletypewriter to interface with it. Since a system of this type met our requirements at an absolute minimum cost, it was decided that this was the way to go. Now there was another important decision to make: What minicomputer system should the school purchase?

That summer, MITS Inc. was running a sale on its Altair 8800 computer. What it offered was the Altair 8800 computer, two 4 K dynamic memory boards, an interface board, and, most importantly, their 8 K version of BASIC on paper tape, all for only $995. This meant that the system would pay for itself in less than a year, as compared with the next most expensive alternative. Table 1 shows the breakdown of costs we estimated during the summer of 1975.

The Teletype Model 33 ASR was selected for use as the terminal for several reasons: It provides hardcopy output, it has a paper punch and reader, it does not need a telephone connection; and we knew from previous experience that it is rugged and reliable, with maintenance, as needed, readily available under the leasing agreement.

Table 1: Comparision of Two Year Computing Costs.

SYSTEM	TOTAL COST
IBM 5100	*$9000*
Wang 2200	*$5400*
Commercial timesharing	*$6740*
Phone line cost – $150 per month	
Computing costs – $100 per month	
Terminal with dialup – $87 per month	
Altair Package	*$2195*
MITS Altair plus software – $995	
Teletype Model 33 ASR – $60 per month	

Assembling the Altair

A check for $995 was subsequently mailed off to MITS in New Mexico, and we waited for the kit to come . . . and waited . . . and waited. After almost two months of patience, the kit arrived at the school in late October. Since my father and I were charged with actually building the thing, I had to bring the kit (data bus and all) home with me on the school bus (which was an experience in itself)!

The assembly manual for the Altair was somewhat disappointing in its handling of errata information. When MITS makes a change in one of the kits, it throws a pile of modification and errata sheets into the front of the manual. While the information is complete, this makes it hard to keep up with the changes that have been made. A better solution might have been to issue replacement "change pages" to be substituted for uncorrected originals. Another minor disappointment was the fact that not all the bugs had been caught by MITS. One such uncorrected mistake was the fact that the "+" and "–" signs on the power supply's bridge rectifier did not line up with the corresponding signs printed on the board itself. My father and I ended up having to trace the proper connections on the schematic to see what the correct alignment was. I believe that anyone who was unfamiliar with working from a schematic would have some trouble understanding how to orient that rectifier.

Other small problems included nuts, bolts, and screws that always seemed to be the wrong size for the job, and a shortage of terminal lugs.

Working nights and weekends, my father and I completed construction within two weeks. Powering up the kit for the first time, we discovered that the only defective part was one LED on the front display panel. The only thing left to assemble was the serial IO board. This time the assembly instructions were clear enough, but the theory of operation manual was somewhat sketchy.

One thing that MITS failed to mention was how to program the Altair to talk to a Teletype! You would think that they would mention that the interface must be set for 8 data bits, no parity bit, two stop bits, and device addresses 000 and 001, right? Wrong! This information was not mentioned in the documentation. Apparently MITS cannot tell you how to interface the Altair with any specific terminal because they have no way of knowing what kind of device you would be using in the first place. It is fortunate that we had read Don Lancaster's article on serial interfaces in the September 1975 BYTE. My recommendation on this point would be a set of examples showing several typical cases.

Up and Running

Finally, after a long delay in obtaining the Teletype (not purchased from MITS, but leased from RCA in New Jersey), the system was fully operational. We have been using it continuously ever since.

MITS is to be congratulated for their excellent software. Their version of BASIC is superior to most others that we have encountered, and it uses only 6 K of memory, allowing us to write programs of considerable length (about 100 lines).

The Altair is kept powered up continuously from Monday morning to Friday afternoon to save wear and tear on the paper tape with the BASIC software; also, it would be too inconvenient to key in the bootstrap program and wait the 12 or so minutes it takes to load BASIC every day.

The security of the system is important because the Altair and the Teletype are both kept out in the same classroom. Because the computer is not very large and thus easy to steal, special precautions had to be taken. The Altair is attached to the cabinet by three screws through the bottom of its case; it is positioned close to the rear of the cabinet so the top of its case cannot be removed with a regular screwdriver. Also, a Plexiglas shield was placed over the bottom of the Altair's front panel to keep anyone from inadvertently throwing the "OFF" or "RESET" switches and wiping out the BASIC.

A typical session at the terminal goes something like this: The student connects the computer and the Teletype by turning the selector switch to COM (communicate). There is, at present, no sign on or user password to control access to the computer. Making the student use the computer out in the open discourages those who are not authorized to use it from doing so. (Besides, 99% of the people who aren't supposed to use it wouldn't know what to do even if they got it powered up!) After making the connection, the student clears the memory to remove any data that may have been left by a previous user. He then either keys in his program or loads a program through the tape reader, and goes to work. He may also use a program from a library of special routines we have in several subject areas: mathematics, chemistry, and physics problem solving, lab simulations, text editing, and puzzles and games. When his session is finished, he can save the current program on paper tape, or simply clear the memory and turn off the Teletype.

Future Plans

What does the future hold for this system? The first addition planned for it is the conversion to a magnetic tape interface. The MITS cost for this interface plus an additional expander board and a cooling fan comes to less than $170. The conversion will accomplish three important things: It will shorten the time needed to load BASIC from 12 to four minutes, provide for program storage in the more convenient form of tape cassettes, and it will allow us to trade down from the Model 33 ASR to the 33 KSR. The advantage of trading down is that the KSR leases for $15 per month less than the ASR since it does not have a paper tape punch or reader. The savings will pay the cost of the cassette interface in about a year.

Further in the future, we see the memory expanding to 12 K bytes, purchasing the Altair Floppy Disk System, and trading up to MITS Extended BASIC language, which has double precision arithmetic, controlled format output, and disk files. A TV typewriter or other similar video terminal is also envisioned.

Although this article has focused on use of a kit computer as an economical system for a small high school with a tight budget, the savings outlined are applicable for schools anywhere. In the face of rising commercial computing costs, a homebuilt minicomputer such as the Altair offers an economical yet efficient alternative to commercial computing systems for schools.

This article first appeared in the June'76 issue of BYTE magazine; it is reprinted with permission of the author and the nice people at BYTE.

AN UPDATE FROM CHRISTOPHER LETT

The Kennedy High School Altair has been in operation for about a year now. In that time, the following changes have been made:

(1) A cooling fan has been installed, as the computer is left powered-up for weeks at a time.

(2) The memory has been enlarged from 8 K to 12 K to handle larger programs – the 2366 bytes free in the 8 K setup were sometimes inadequate for larger programs. The additional memory is a MITS 4 K Static. Unfortunately, it loaded down the Altair power supply to the point that operation became unreliable. The MITS Power Supply update was inadequate, causing gradual memory errors. We added a 0.01 μf capacitor across the zener diode, making the power supply functionally reliable but clearly inadequate for the total load planned. We now plan to upgrade the power supply *correctly* with the $75 kit from Parasitic Engineering. This will let us run reliably at full capacity.

If we were to obtain equipment now, we would definitely get on IMSAI or an Altair 8800b, with adequate power for 16K of memory. The standard Altair power supply is unreliable in a system where reliability is paramount.

(3) The Teletype ASR-33 will go off rental in February. The following will replace it

- **VDM-1 Video Display Module from Processor Technology will be connected with a Hitachi PO-5 12 inch black/white TV Set (converted with Pickles & Trout TVM-04 kit to a TV Monitor) for soft-copy assignments.**
- **SWTP PR-40 Alphanumeric Printer for 40 chars/line of 64 uppercase ASCII symbols (full BASIC character set) for hardcopy output of class assignments.**
- **Homebuilt Audio Cassette Interface to allow each student their own cassette for their programs.**
- **Homebuilt Keyboard I/O and an ASR-33 compatible keyboard for typing in programs.**

The total cost is about $700, which is less than the $840 per year for rental of the ASR-33. When installed and operational, the computer system will be completely owned by the school. It is currently being used by approximately 60 students in a BASIC language programming course.

After all this additional hardware is integrated into the system and operating reliably, extension to other new applications will be resumed.

Anyone interested in learning more about the details of the system can write me:

Christopher Lett
123 Crockett Hall
RPI
Troy, NY 12181

CABRILLO COMPUTER CENTER

SUMMARY OF ACTIVITIES by HAL SINGER

Cabrillo Computer Center
Cabrillo High School
4350 Constellation Road
Lompoc, CA 93436
805-733-3501

PRESENT EQUIPMENT

1 each	12K PDP-8/E with LA36 DECWRITER 32K fixed head disk mark sense card reader punch card reader homemade paper tape reader and punch 64 x 64 dot refresh graphics terminal & light pen squawk box (computer controlled amplifier)
3 each	DEC classics consisting of: 16K PDP-8/A dual floppy disks VT50 video terminal

ADDITIONAL TERMINALS

2 each	LA36 DECWRITERS
2 each	ASR-33 teletypes
1 each	Centronics 508 incremental line printer with keyboard

ADDITIONS BEING WORKED ON

dual digital cassette system
mini-magtape system (emulates DECTAPE)
TD8/E DECTAPE system (need to acquire or make controller)
120 cps papertape punch
650 cps papertape reader
256 x 256 dot matrix refresh graphics terminals
vector graphics terminal
Selectric typewriter output terminal
16 x 32 TV typewriter
analog x-y plotter

COMPUTER CONSTRUCTION PROJECTS

1) Mike Ceruti is assembling as a school project an S100 bus machine to include the following components:
   A) Morrow Micro Stuff 8080 CPU board
   B) 12K SD Sales low power RAM
   C) 4K Solid State Music PROM
   D) Solid State Music 16 connector mother board
   E) 2 ea Solid State Music I/O boards
   F) homemade power supply
   G) 16 x 32 Suding TV typewriter
   H) dual Digital Group Phideck system
   I) 40 column 120 cps Monroe printer
2) Mark Burnett is assembling as a school project an Altair 680. Future expansion plans are still open but will probably include a mother board bus converter to allow S100 peripherals, especially memory to be used. A Creed Baudot TTY may be used for I/O and a Byte Standard cassette tape interface will be added.
3) Jeff Fint and Jerry Nix are assembling an IMSAI 8080 for John Craig, editor of *Kilobaud* magazine.
4) Dave Bryant is completing a full feature 8080 Phideck operating system that will include full directory oriented saving and retrieval of programs.

PROJECTS ON WORKBENCH BUT LOW ON THE PRIORITY LIST AT PRESENT

1) $100 Glass TTY design featuring:
   16 line x 40 character display
   current loop I/O
   full scrolling
   uses SC/MP for intelligent control
2) SC/MP prototyping system featuring 8K RAM, 4K PROM
3) Martin Research Mike 2 System (8008)
4) Mark 8 System (8008)

EDUCATIONAL COMPUTER PROGRAM

Computer programming courses offered:

1) Beginning Computer Programming
   (1 semester)
   The student should be a skilled "basic" programming language programmer at the completion of this course. Bob Albrecht's books "Teach Yourself Basic, I and II" are used.
2) Intermediate Computer Programming
   (1 semester)
   Features programming in FORTRAN. Uses textbook "Introduction to Computer Science" by Adams and Moon
3) Advanced Computer Programming
   (1 semester)
   Features an introduction to computer science and PDP-8 assembly language programming.
4) Advanced Computer Programming II
   (1 semester - can be repeated for credit)
   A project oriented course for advanced students.

Usage plans require utilization of the equipment in other educational areas. Present usage consists of:

1) math fundamentals drill and practice
2) chemistry drill, experiment calculation, verification
3) physical science laboratory preparation
4) economics stock market simulation.

Future plans consist of expanding educational uses to all academic areas and doing limited administrative service jobs.

MAJOR SOFTWARE DEVELOPMENTS

1) EDU25 time share BASIC modifications to allow floppy disk program and file storage. Written under contract by Dave Bryant. Contact the computer center for information.
2) FILSAV – an OS-8 service program that allows one to save ASCII files on floppy disks with crashed directories. Written by Dave Bryant.
3) Football scouting report systems – two very elaborate systems used by the high school football coaches to analyze opposing team tendencies based on analysis of play information. Each system produces about 5 feet of detailed analysis printout. We have been told that the systems compare favorably with those being used by the professional teams. Contact the computer center for information. □

HAVE MICRO, WILL TRAVEL

by Joanne Koltnow Verplank

COMMUNITY COMPUTER CENTER

This article is designed to answer the 'How Do I Do It' questions of people who would like to share their computers with others. It will also hopefully spark the imaginations of those readers who had not previously considered this use for their computers. The idea for such an article has been perking for some time and was finally concretized after my visit to a meeting of the Southern California Computer Society (SCCS).

For the past year or so, I have felt that personally-owned computers were no more than expensive toys and that people were trying to justify the expense when they talked about the practical uses to which their computers could be put. Whether in fact they are used for entertainment or for other purposes, personal computers are certainly gaining in popularity. Clubs have developed and publications have been started which help the increasing number of computer owners to communicate with one another. This communication and its consequent sharing of knowledge are important if we are not to have countless individuals each solving the same problems that others have already finished. The clubs and publications provide a means by which computer owners can share their experiences with other owners. What is needed now is a way for owners to share their *computers* with non-owners. I suggest that the sharing will give satisfaction to both parties in the arrangement, and that it will be relatively easy to accomplish.

Some background: I work at the Community Computer Center, a place dedicated to sharing computer power. We teach classes and generally make our computer available to the public. We have been doing this for four years, and while we're still struggling financially, we have had a great deal of experience in acquainting people with computers. The visitors to the center (mostly children) are excited by what's available, and we enjoy teaching and experimenting with new ideas.

The computer center is a unique place, and it's easy to forget that sharing one's own computer is harder for an individual than it is for us (at this point in our existence). After talking with some people at the SCCS meeting, I got an insight into the problems facing potential computer sharers. The advice 'Take your computer into a school or library' is really inadequate for someone who is inexperienced in working with children Keeping a group of children relatively quiet and interested, while teaching them something, is no small task for the novice.

In the rest of this paper, I will outline things to say to get yourself and your computer installed in a school or library. More important, I will tell you just what to do when you get there. These suggestions are culled from several years of teaching at the computer center as well as from teaching in public schools before that. They serve a dual purpose: to intrigue more people into sharing their computers and to fulfill my promise to provide help for people who have already decided to take their computers into the community.

I will discuss taking computers to schools and to libraries. The category 'libraries' can be expanded to include community centers and Y's. If you like structure, schools are the easiest to work in. In schools, the students are disciplined already, and the teacher will probably handle the logistics. Schools have the inconvenience of being open only during the hours when most people are working, however, so you may have to settle for one of the other locations.

Let's consider schools first: Many teachers will be happy for you to bring something educational into the classroom and to work with small groups of children on a regular basis. The key words here are *educational* and *regular.* Teachers are interested in education and in the dependability of their volunteers. It will be helpful if you schedule a regular time for your visits so they can be worked into the classroom schedule. You may also find takers for a one-shot demo, if you're not sure you want to commit yourself to regular visits. The same suggestions will apply as for a series.

Consider the educational value of computers – your computer in the flesh as opposed to an abstract concept:

1. *New experiences are educational.* Seeing a computer close up and working with it fits nicely here.
2. *Computers are increasingly part of our lives.* Your visits may become part of a unit on computers, or on modern technology. You may become a resource person for class projects.
3. *Computers can perform operations very quickly, but they have to be instructed precisely.* This can relate to language and math curricula. You may decide to show some examples of how imprecise thinking can lead to an unexpected result in a program.
4. *Computer games/exercises are available that strengthen logical thinking, teach specific concepts, and develop skills in various areas.* I'm appending a chart relating some computer games and curriculum areas.

Use these arguments, and any others you may think of, and be prepared to show the teacher a sample of what you'll be doing. The sample may not be necessary, but it will probably transform the teacher into an active fan. The games/exercises we use at the center really are fun for the players and also help develop and strengthen skills that educators are interested in. In fact, the teacher may see classroom applications that you had not thought of.

Once you're in the classroom: Tell the teacher that you'll work with a small group of children. The group size will depend on the kind of display your terminal has. It will also be affected by how comfortable you are with groups of youngsters. You'll want to have few enough to keep them interested and relatively quiet, but enough for an interaction among them and a discussion of their strategies. Three to five is a comfortable group size.

For simplicity, I operate the terminal when I'm teaching a game or when I'm working with a large group. This lets the students concentrate on learning and playing. They can sit back from the terminal and you don't have to worry about their pushing one another in their enthusiasm, or fumbling with the keys while others are waiting impatiently. Players don't have to press the keys to be involved in the game. There is an advantage, however, in being in a *small* group for decision-making, so I divide large groups into smaller teams.

If you have a program that is totally do-it-yourself, you can fade into the background. The players will relate to the game instead of to you, and they will learn to be precise in their responses so the games will go smoothly. This will also lead to their becoming self-sufficient, one of the more general aims of education. STARS is a good beginning game for any age group. The computer's responses are visual rather than verbal, and STARS can be played successfully by beginners, while advanced players take the challenge of breaking the code.

We want players to be able to stop a game easily. Most of our games respond to the following words:

HELP – which prints directions
RANGE – which lets the players change the difficulty level
STOP – which stops the game
QUIT – which gives the answer

The incorporation of these key words into our games has made running the center and teaching classes much easier.

Some of the messages and error traps we have developed for our games are displayed in the following runs of STARS.

```
*** CCC GAMES MONITOR ***

GAME? STARS

I HAVE AN INTEGER FROM 1 TO 25

YOUR GUESS? HELP
I HAVE AN INTEGER FROM 1 TO 25
YOU ARE TRYING TO GUESS IT.
AFTER EACH GUESS, I WILL GIVE YOU STARS(*).
THE MORE *S YOU GET, THE CLOSER YOU ARE.

TO STOP PLAYING TYPE STOP
FOR A LIST OF SPECIAL WORDS TYPE WORDS

YOUR GUESS? 21
* *
```

```
YOUR GUESS? 12
* * * * *

YOUR GUESS? 9
* * *

YOUR GUESS? 11
* * * *

YOUR GUESS? 13
* * * * * * * * * * * * * * * * * * * * * * * * *
*** YOU GOT IT IN 5 TURNS ***

PLAY AGAIN? YES

I HAVE AN INTEGER FROM 1 TO 25

YOUR GUESS? 34
PLEASE TYPE AN INTEGER FROM 1 TO 25
TO GET HELP TYPE HELP

YOUR GUESS? L
PLEASE TYPE AN INTEGER FROM 1 TO 25
TO GET HELP TYPE HELP

YOUR GUESS? QUIT
MY NUMBER WAS 17

PLAY AGAIN? NO
THANKS FOR PLAYING.
```

One of the recent changes we made at the center was setting the base level in all the games rather low. That way we can be sure that beginners can have a comfortable level on which to explore. More advanced players can change the difficulty of the game. Our philosophy in teaching is: make the exercise easy to learn, then teach people how to set the difficulty level themselves.

```
GAME? STARS

I HAVE AN INTEGER FROM 1 TO 25

YOUR GUESS? RANGE
MINIMUM? 20
MAXIMUM? 40

I HAVE AN INTEGER FROM 20 TO 40

YOUR GUESS? 24
```

From STARS, you can go different ways, depending on the time you have and the capabilities of your computer. Install yourself in the school library or learning center and expose the whole school to your micro via STARS. Or, at the other exteme, take one group in the class and go as far with them as they can go, before you take on another group. Your time and the students' responses will help you decide.

I usually first teach simple games requiring the student to guess a number (NUMBER, STARS) or a word (SNAKE). Our game STORY is also good for beginners: the student answers some questions and then a personalized story is printed out. I follow these with more sophisticated guessing games such as BAGELS (the object is to guess a 2-digit number) and JOTTO (you are told how many letters in your guess are in a target word).

Next come strategy games such as HURKLE (you try to locate a hurkle hiding on a 6 by 6 grid) and WUMPUS (hunt the wumpus through a series of caves that must be explored while avoiding nasty dangers). If the students understand multiplication and factors, the game TAXMAN can also be introduced at this stage.

You can use this series with almost any age group if you pace it to their learning rate. Introduce a few games at a time (maybe just one) and encourage the players to explore until they're comfortable with the game and can operate on various difficulty levels. This method avoids having them try all the games at once, then get 'bored' and want more because they think they know them all. If players get in the habit of exploring each new set of games, you'll better know their capabilities, and they'll ultimately gain more from the experience.

Students who are between turns at the computer can take the computer's part and play the games with another group of students. Practice this with them, so you can see that they understand the computer's role in various games. With young children, I usually take the part of the computer first. We often begin with NUMBER. I tell them that I'm going to think of a number then they'll try to guess it. I give them the same clues the computer uses (try bigger, try smaller) and I count the turns it takes them to guess my number. I remind them that the computer doesn't make mistakes when it gives the clues, so they should be very careful. I also remind them that once the computer picks the number, it doesn't change its mind, even if someone guesses it on the first try.

If you're this far along, you've obviously been successful in managing the programs, and the students are happy. You might at this point consider introducing simulations, or teaching programming. Keep the teacher informed of your progress and of the educational benefits of what you're doing.

Libraries: If you can't get away from work to go to a school, there's always the library. Here, the organization will probably be left up to you, and you might want to consider all the possibilities before you begin. You can bring your computer and let whoever wants to, play with it. You might also teach groups, or choose something in between. Pick a method you'll enjoy and feel comfortable doing. You might go a few times before word gets around, but soon you'll have a crowd. Decide how you want to manage the computer vultures – those people who monopolize the terminal. Maybe set a time limit, or simply have the same game and expect that people will play for a time and then move on. Perhaps you can start a club for the vultures. They might turn out to be your best helpers.

The library is an especially good setting for the 'do-it-yourself' program. You can fade into the bookshelves and watch people read the instructions and try the computer. It's a good idea to explain the educational aspect of this experience, just as you would with a teacher. Who knows what will come of it. Maybe the library will make a special exhibit, so people can read about your computer and its predecessors. Perhaps the local paper will write up your visits to the library or the school. An article will be good publicity for them and will help encourage others who want to take their micro into the public eye.

Last word: Someone said that he had trouble with people trying to crash his system. My advice was to try to avoid such individuals. If it's a high school group, go to the lower grades; if it's a certain group at the library, restrict the access to your computer. I suspect the incidence of this will be slight if you're sure of what you want to offer. Present your computer for a specific use rather than just to be played with in general. Remember, you're sharing it, but if you're unhappy with the use it's being put to, you can always look for a more appreciative audience. Oh yes – the games I talk about are available inexpensively, on paper tape from CCC. They're written in various versions of BASIC, and details are listed in the Bookstore Catalog published with the Nov-Dec '76 PCC, Volume 5, Number 3.

RELATION BETWEEN CURRICULUM AREAS and SOME COMPUTER GAMES

CURRICULUM AREAS	COMPUTER GAMES											
	Animal	Bagels	Cricket	Hurkle	Jotto	Letter	Number	Snake	Starts	Story	Taxman	Wumpus
Numeration – greater than, less than			•	•		•	•		•			
Estimation, prediction		•	•	•	•	•	•		•			•
Logical reasoning	•	•	•	•	•	•	•	•	•		•	•
Sequencing/pattern finding	•	•		•	•	•	•	•	•			•
Alphabetizing						•						
Reading	•									•	•	•
Spelling	•				•			•		•		
Strategy formation		•	•	•	•	•	•	•	•		•	•
Problem solving		•	•	•	•	•	•	•	•		•	•
Multiplication (factors)											•	
Recording data		•	•	•	•	•	•		•			•
Following written directions	•	•	•	•	•					•	•	•
Writing sequences	•											

FORTRAN MAN

·BY LEE SCHNEIDER AND TODD VOROS·

As you may have noticed from previous episodes, the normally tranquil and passive 'Old Country' of Transistoria has become incredibly active for the past few cycles . . . First, there was the arrival of the evil Count Algol, and terror spread through the land . . Then, the one and only Compiling Crusader, Fortran Man, arrived to do battle with the evil nemesis! The first encounter of the two super-foes proves nearly fatal to F-Man - as the villagers watch in horror, Our Hero is flung from the highest level of the Count's castle!! Any normal compiler would have been doomed . . . but the resourceful F-Man manages to survive the near-fatal systems crash . . .

F-Man pauses only long enough to allow his old friend Doktor Debug to reset his fractured code before he and the Doktor (and the Doktor's beautiful daughter Parity) set out for Junction City to seek aid from . . . the Monitor! The journey is long, and Parity is almost lost in the transmission, but at last they stand before the office of Transistoria's most powerful and illustrious citizen, Ludwig von Monitor!!!

Still wearing the software patches applied by Doktor Debug, F-Man ascends the stairs to the high-level code area, and despite the warnings of the Doktor, he rushes past the surprised service routines and branches directly into the Monitor's office!

Hold on a few usec, F-Man . . . you shouldn't go interrupting the Monitor without a formal request . . . especially in your condition!

Don't worry, old friend . . . this will only take a millisecond or two!

IN
OUT
NOP

Sir! Wait! You can't . . .!

Within the large office, F-Man is confronted with a myriad of control blocks and vectors, and, among them, the incredibly busy ruler of Transistoria – Ludwig von Monitor himself!

Excuse me, sir . . . .

I have an urgent matter to discuss with . . . .

Vat ist this! Ein unprivileged job in mein office??!!!

Vere are your papers??

Vere ist your priority number??!!

The resourceful Fortran Man can generally communicate as well as any other software in the system, but in his current condition, he seems to find it difficult to get a byte in edgewise . . . .

You have 15 microseconds to explain yourself!

er . . . uh . . . .

Time ist up!!!

The time limit expired, the Monitor activates one of his many control lines, and . . . .

ding!

DING!

INTERRUPT

click!

ACK!

zip!

Meanwhile, the Doktor is leisurely relocating himself to a lower-level area . . . .

LOW LEVEL CODE AREA

Come on, Parity dear . . . . we'll wait for F-Man down here!

But father – the Monitor's office is on the upper level!

Yes, I know – but I also know old Ludwig . . . . and if my computations are correct, F-Man should be along just about . . . .

DMA ONLY!
MASS
STORAGE

KLONG!

. . . . . now!

ZIP!

Fortran Man, however, is not so easily defeated . . . . and, though slightly out of phase from his recent rough treatment, he determinedly resets his pointers and branches directly for the Monitor's office once again!

Er, just a moment, F-Man! I think there's something you ought to . . . .

Not now, old friend! This is a matter of great importance, and I intend to see it done!

A few moments later, Fortran Man branches back into the office and begins pass 2 at the Monitor . . . .

Sorry to keep bothering you, sir, but this is a matter of vital urgency!

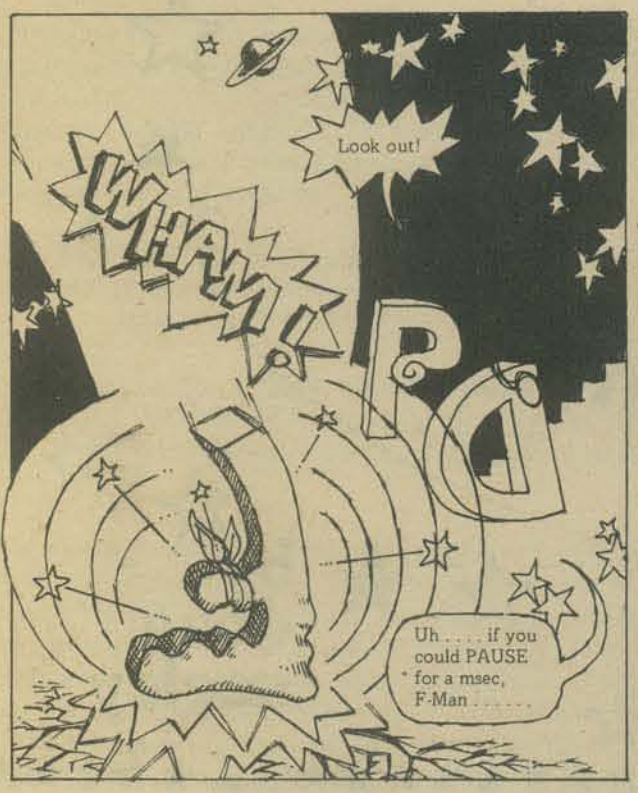
VAT IST??

YOU AGAIN!!!!!

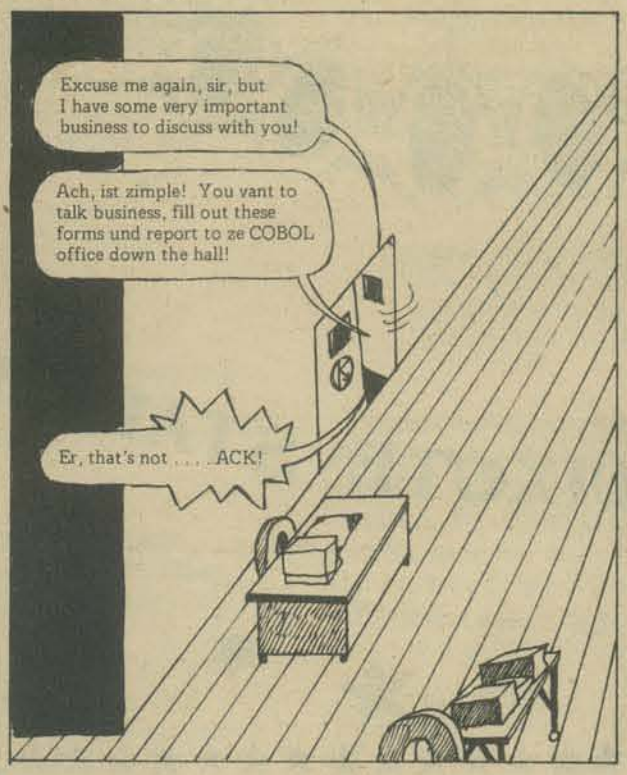
Now if you'll just give me a moment to explain . . . .

· DRAWING BY A. MIYA ·

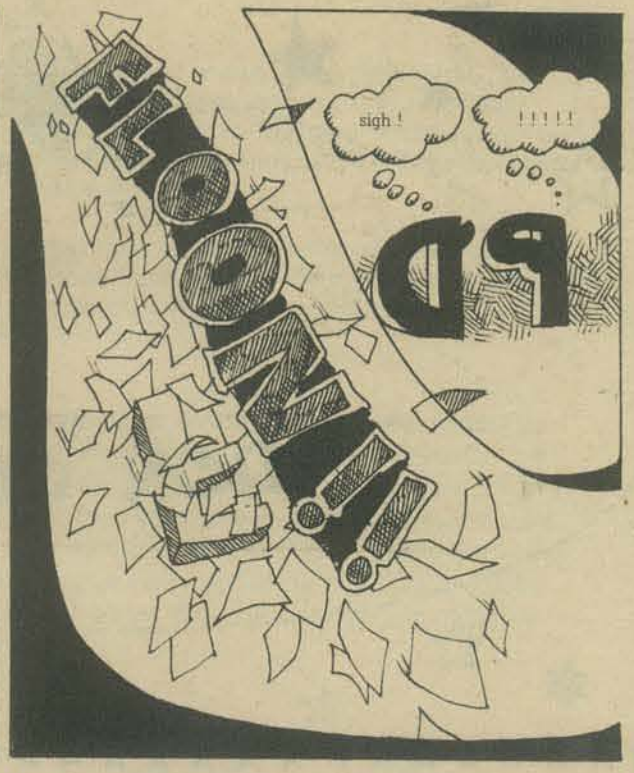
And in yet another moment, pass 2 is terminated abruptly . . . .



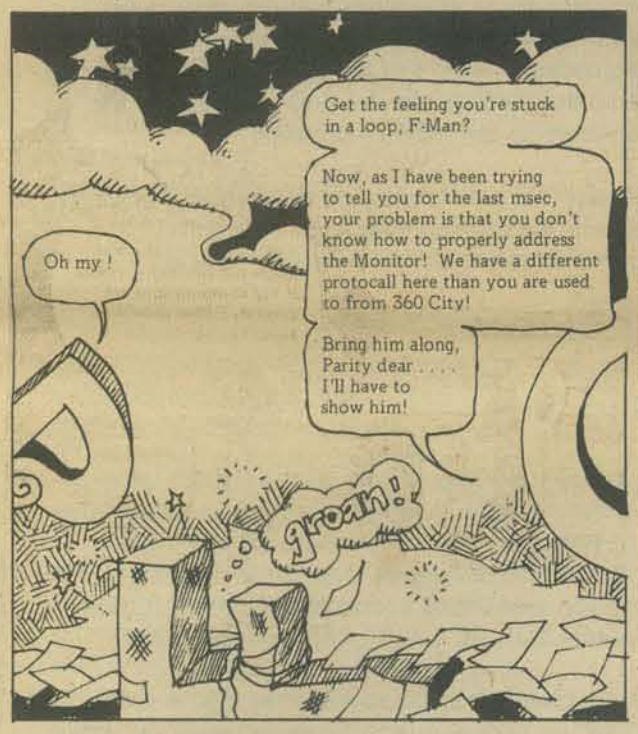
Frustrated by this unprecedented lack of success, F-Man gives it one more try . . . . .



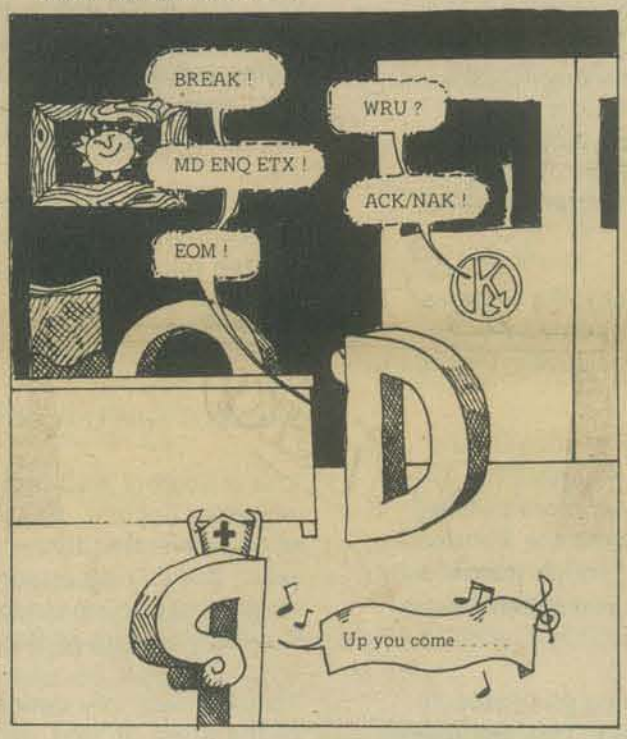
There is a sudden form feed, and then, once again . . . . .



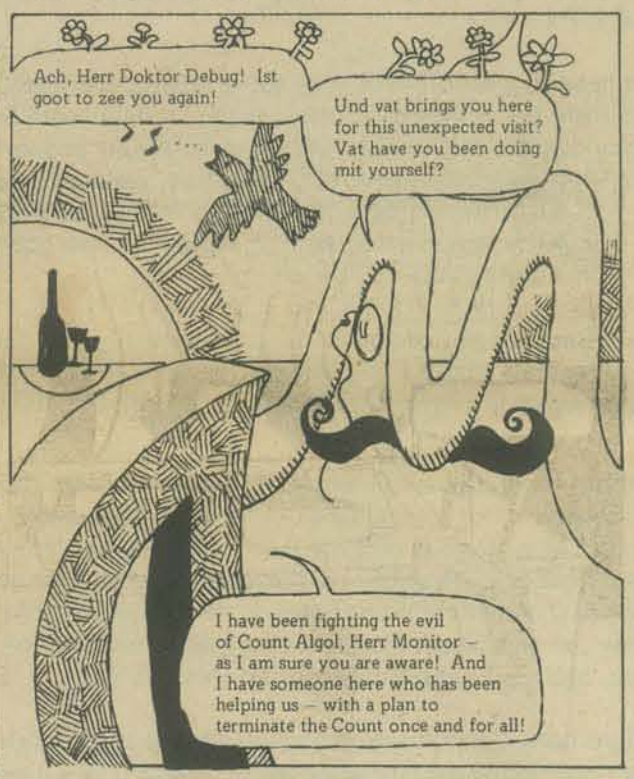
Even a superhero such as F-Man has a limit, though . . . . . and after three successive file dumps by the Monitor, Our Hero is in sad shape . . . . .



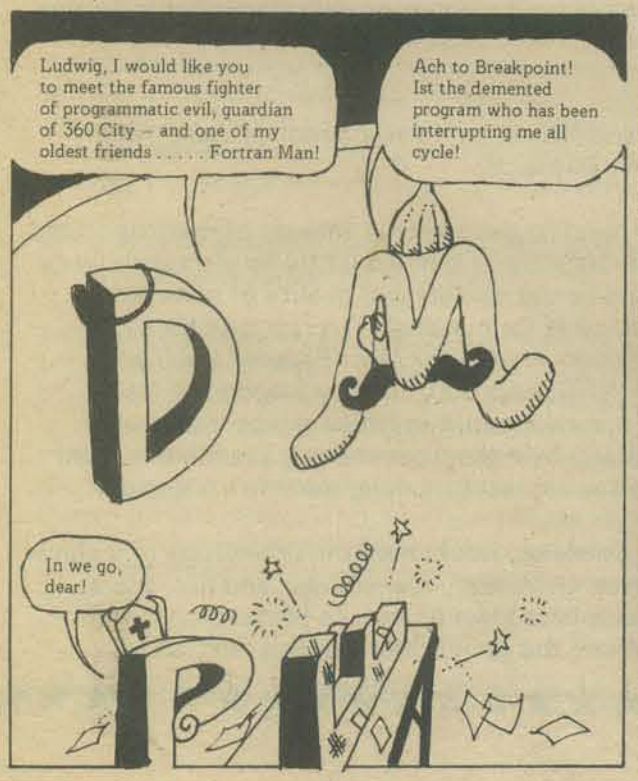
As Parity assists the weary F-Man back up the level shifter, the good Doktor demonstrates the correct way to interrupt a Transistorian Monitor . . . . . .



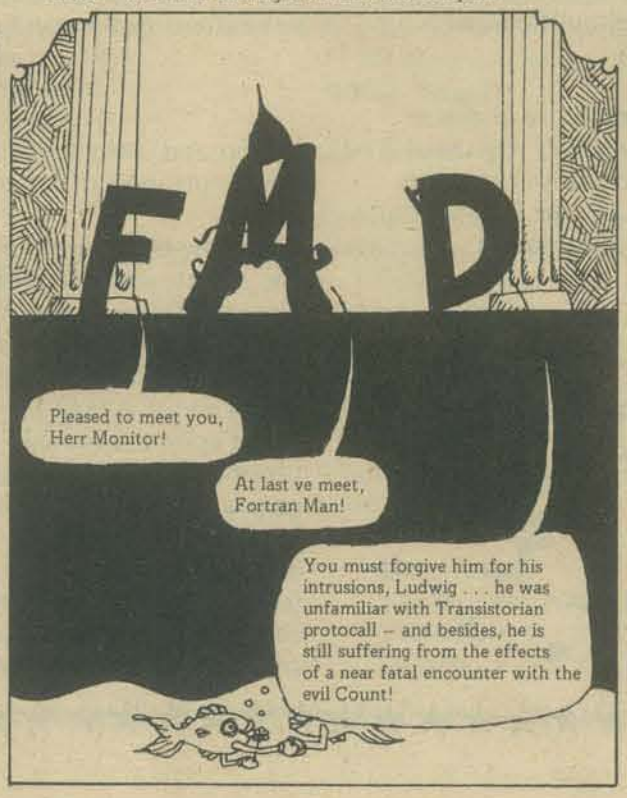
Acknowledgments completed, the Doktor steps into the office . . . . .



The good Doktor signals to Parity and F-Man who are waiting outside, and . . . .

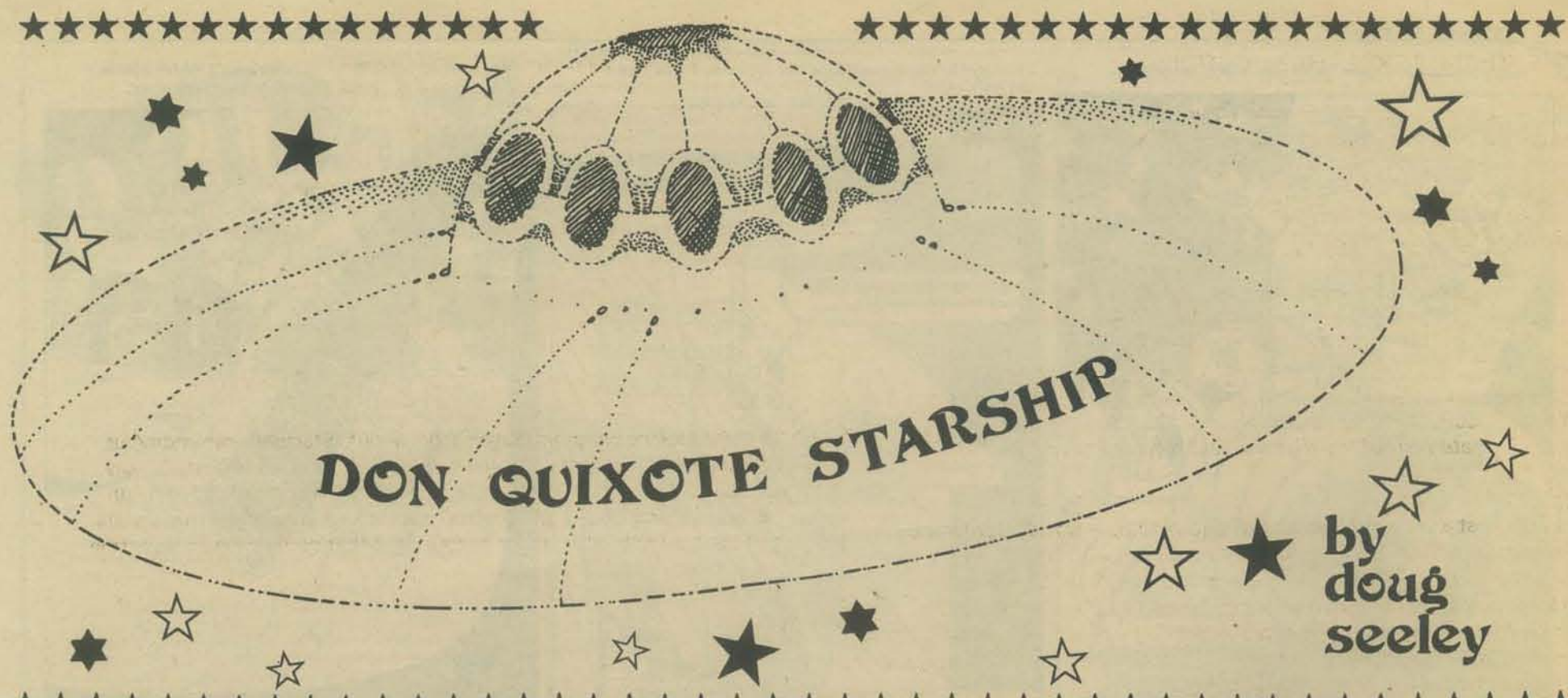


At last – the formal meeting of Monitor and Compiler!



Finally, the great meeting commences . . . . .

What is Fortran Man's secret plan for terminating the Count? Will Transistoria ever be free again? See you again next episode – same time, same software . . . . . .



HELP!!!

We need suggestions, designs , experiments, software, etc., that will facilitate the computer gaming of the Don Quixote Starship (DQS). Amongst others, the game will have two basic properties; it will be *distributed* and *co-operative*. Therefore, we need to develop

1. a distributed data-base system for microcomputer users
2. a micro net: an interlaced system of microcomputer networks
3. a global network

Data Base — Let's design this around cassette tape decks; the floppies have formats that are too varied. However, one could come up with a somewhat universal data-base interface that would be device dependent.

Should the data-base be organized especially for DQS or should it be somewhat more general like Community Memory?? An overview of some CM systems is on Page 9 of this issue, but let's get more detailed specifications. How can we network the data-base across the Continent: should there be collector-bases dotted conveniently through the network? How can this be decided and operated? Have any of you games-by-mail folk, such as Flying Buffalo, tried this out??

Micro nets — Are there computerfolk out there working on protocols for networking micro-computers?? Many things need to be specified for them: hand-shaking routines, routing algorithms, message sequencing, error-detection codes, high-speed acknowledgments, simplified user-interfaces, etc.

Perhaps some folk have tried to link their micros together by phone or cable; has anyone done this in a public and accessible way?? By that I mean, has anyone developed a micronet that is not special-purpose but allows general exchange of messages (i.e., that comprise user interactions, information, and programs)?? How can a micronet facilitate the distributed data-base for the DQS??

Global Networking — Has anyone tried telemetry interfaces with micros yet? The Aloha computer network in Hawaii uses short-wave radio links. Could we set up a link between two remote micros using ham radios and error-correcting codes? Using the codes and low transmission rates, such links could percolate throughout the night and across the world, bringing in daily updates to the DQS data-base!!

There has been recent interest in a public satellite that would, amongst other activities, facilitate radio hobbyists. Can such a situation be utilized for computer communications by hobbyists in ways such as described above?

PROJECT DAEDALUS!!!

This is another seed project by the intrepid folk of the British Interplanetary Society. Originated by Alan Bond, it is a venture to perform an undecelerated fly-by of Barnard's Star. At a distance of 6 light years, this star is currently thought to have at least three planets. Its motions have been studied for over thirty years. See 'Concerning the Planetary System of Barnard's Star', ICARUS, 1973, pages 353-373.

They are seriously designing a nuclear pulse rocket that would reach 14% of the speed of light, and would take about forty-seven years; for example, see *CETI*, by Jack Stonely and A. T. Lawton. Several space simulations/games can be designed around considerations of this project.

Players and designers of such games could even produce useful results for the planners of Project Daedalus.

One game that comes to mind is, given a fixed amount of mass/fuel/food etc., construct interstellar trajectories that would fly-by stars with likely planets. The results would be the number and quality of surveyed planets that would be habitable for humans! The raw data for such a game are available in *Habitable Planets for Man* (Blaisdell Publishing, 1964) and ICARUS 13 (1970), page 500, both by Stephen H. Dole. Based upon the probabilities he establishes (these can be modified optimistically, or pessimistically by a game parameter), a random number generator could establish the contact-hits along the ship's trajectory.

Such a game, let's call it Daedalus, would have the side-effects of a player learning celestial mechanics, elementary planetology, and near-space cosmography. Does anyone have ideas on how to bring co-operation and perhaps competition into the game? More on this next issue.

THE COSMIC CONNECTION: CETI !!!

*C*ommunication with *E*xtra-*T*errestrial *I*ntelligence, CETI (pronounced 'setty') is now a legitimate concern of the scientific establishment in the West, thanks to Carl Sagan, A. G. W. Cameron, and the enthusiastic Russians. Several books on CETI (sometimes called exobiology) have been published in the last fifteen years, and especially recently. For instance, Cameron's books:

The Search for Extraterrestrial Life, W.A. Benjamin (1963)
Interstellar Communication: Scientific Perspectives,
Houghton Mifflin (1974)

Two recent books by Sagan have provided a wealth of information and current debate. The first, edited by him, is called *Communications With Extraterrestrial Intelligence,* and is published by the M.I.T. Press.

It is almost a verbatim account of a joint U.S. – Soviet Conference on the subject convened in Soviet Armenia in September, 1971. What is particularly significant is the inter-disciplinary scope and quality of the participants. Amongst well-known astronomers there was Francis Crick of double-helix fame; David Hubel, a novel experimenter on the neurophysiology of the visual system; Marvin Minsky, a leading light of the Artificial Intelligentsia; and Freeman Dyson, the far-out ideaman of Princeton's Institute of Advanced Studies. Some of the topics included:

Extrasolar Planetary Systems
Extraterrestrial Life
The Evolution of Intelligence
The Evolution of Technical Civilizations
The Numbers of Advanced Galactic Civilizations
Astroengineering
Techniques of Contact

Of the several conclusions, the following is telling:

> This problem may prove to be of profound significance for the future development of mankind. If extraterrestrial civilizations are ever discovered the effect on human scientific and technological capabilities will be immense, and the discovery can positively influence the whole future of man. The practical and philosophical significance of a successful contact with an extraterrestrial civilization would be so enormous as to justify the expenditure of substantial efforts. The consequences of such a discovery would greatly add to the total of human knowledge.

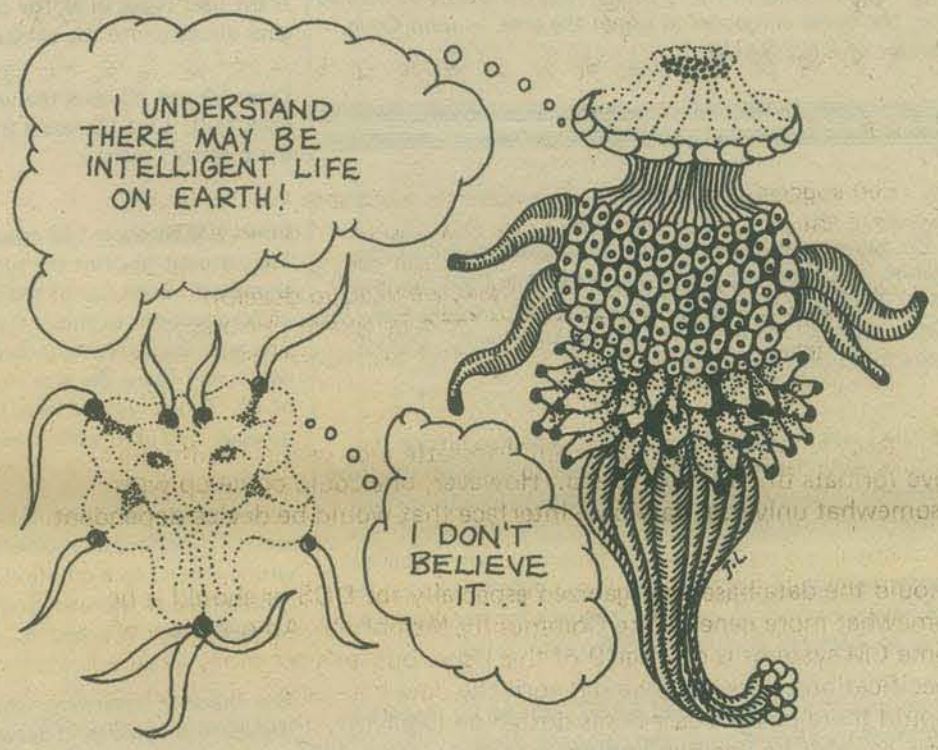
This book is a must for someone interested in the real possibility of galactic exploration and contact with alien intelligence, and for game designers. However, Sagan elaborates upon a covey of provocative ideas that are often outside 'legitimate science'. Consider the following suggestions about 'them' contacting us (Page 224).

> Consider, for example, seashells. Everyone knows the 'sound of the sea' to be heard when putting a seashell to one's ear. It is really the greatly amplified sound of our own blood rushing, we are told. But is this really true? Has this been studied? Has anyone attempted to decode the message being sounded by the seashell? I do not intend this example as literally true, but rather as an allegory. Somewhere on Earth there may be the equivalent of the seashell communictions channel. The message from the stars may be here already, but where?

In another section he captures the emotional yearning of those who would explore the stars (Chapter 32).

> For three generations of human beings there was – as an ever-present but almost unperceived part of their lives – a sound that beckoned, a call that pierced the night, carrying the news that there was a way, not so very difficult, to leave Twin Forks, North Dakota, or Apalachicola, Florida, or Brooklyn, New York. It was the wail of the night freight, as haunting and evocative as the cry of the loon. It was a constant reminder that there were vehicles, devices, which, if boarded, could propel you at high velocity out of your little world into a vaster universe of forests and deserts, seacoasts and cities.
>
> There are as yet no interstellar trains, no machines to get us to the stars. But one day they may be here. We will have constructed them or we will have attracted them.
>
> And then there will once again be the whistle of the night freight. Not the antique sort of whistle, for sound does not carry in interplanetary space or in the emptiness between the stars. But there will be something, perhaps the flash from magnetobrehmstrahlung, as the starship approaches the velocity of light. There will be a sign.
>
> Looking out on a clear night from the continent-sized cities and vast game preserves that may be our future on this planet, youngsters will dream that when they are grown, if they are very lucky, they will catch the night freight to the stars.

In three closing chapters, Sagan talks about 'Starfolk', reminding us that we are all composed of the products of stars, of interstellar gas and dust. He is really detailing a galactic ecology, that includes for instance, the necessity of Neutron stars, the end result of many stars and the probable sources for cosmic rays that contribute to mutation and evolution. He also includes a vast interlacing of space and time throughout existence via black-holes, the presumed ghosts of heavy stars. It is excellent reading!



CO-OP SPACE COLONY

In our next issue we will review Gerald O'Neill's *The High Frontier,* and scour it for ideas for subgames of the Don Quixote Starship. One direction that could be pursued is the planning, design, and financing of a Co-op Space Colony, one built for example, like the eco-farms of the New Alchemists at Wood's Hole, Mass. and Prince Edwards Island, Canada.

ZETA RETICULI CONTROVERSY

One of the most intriguing UFO incidents, the Barney and Betty Hill episode, has led to some exciting *computer exploration* of the stars in a fifty light-year radius. The controversy centers around a so-called map of stellar trade-routes of the home-star of the aliens. It is documented in the *Zeta Reticuli Incident* by Terence Dickinson and the publishers of Astronomy magazine (757 N. Broadway, Suite 204, Milwaukee, WI 53202).

Alien correspondent in charge of editing Don Quixote Starship material:

Doug Seeley
833 W. 18th Avenue
Vancouver, B.C.
Canada

• • •

7 x 9 = 56 ... RIGHT?

BY JACK A. INMAN

Without a doubt one of the most significant applications for the home computer in the years to come will be in the area of education. Hopefully, the computer will contribute a lot toward making the learning process a 'fun process'. I've written a program such as Jack's, and I've experienced the fun of watching my kids light up when working problems with the computer. I'm hoping Jack's article will be the first in a long string by many others showing us how to put the home computer to use in the area. – John Craig, Editor of KILOBAUD.

My son was having quite a bit of trouble with his times tables in school. At about the same time, I got my Southwest Technical Products 6800 Computer running well. My wife was after me with the classical statement. 'Boy, it sure looks nice but what can it do?'

While I was building the machine, I was also hitting the books quite hard trying to learn how to program the 'Iron Monster.' Southwest Technical provides excellent documentation with the machine just as they say in their ads. The machine is ready to accept programs in hexadecimal code the minute it is fired up. But, I didn't like the idea of having to work in machine language for my programming; so, I started studying BASIC.

I purchased a book entitled *BASIC* written by Robert L. Albrecht, LeRoy Finkel, and Jerald R. Brown. I found this to be an excellent self-study text on the subject. I then purchased a copy of Tiny BASIC for my machine and started programming. It was from the book *BASIC* that I got the idea for this program. The authors have a similar program for an addition drill in the text. I modified their program for multiplication. Then to put the icing on the cake, I added some scoring for the user.

I taught my boy to operate the Teletype so that he could use the program, and he really does enjoy running it. For the first time, his learning experience is actually fun. He has been caught several times smiling when he sees that he's come up with the correct answer to a tough questions and I believe that the machine has been very helpful to him as well. He has made great progress – he now knows most of the answers and after only a few rounds of the program.

Before going into the program itself, let me give you some of the details of my system to help you in your evaluation. As I said, I have Tiny BASIC for my machine, however, this program can be adapted to any version of BASIC that the reader may have. Since I have 4K of memory in my machine, when I have both Tiny BASIC and this program in the machine, I have 130 bytes of spare memory left. If the reader feels that this is over-crowding, all of the remarks and even the rules and instructions can be left out. The scoring can also be shortened if one so wishes.

Now, let's analyze the program itself. For those of you who know BASIC this may be old hat, but I am going to try to give enough detail so that hopefully even those who are unfamiliar with the language will understand the program.

Fig. 1 is the complete listing of the program. The program begins by printing what it is. It then asks the user, 'DO YOU WANT THE RULES?' To see the rules, the user can type in a Y or any other letter except N. This answer tells the machine to print out the rules in Lines 20 through 46. If the user types in N, the program skips over the rules and goes directly into the execution of the program.

Lines 60 and 70 reset the values in the scoring registers back to 0. Line 60 resets the G (good) and line 70 resets the E (error) register.

Lines 100 through 140 control printing out the score. They are set to print the score after every 10 questions are answered. Each run of the program will ask a maximum of 50 questions as this is the upper limit set by line 140. The answers can be any combination of right or wrong which will cause the scoring registers to equal a multiple of 10. As we will see later the program loops through 110 through 140 after each answer by the user.

Line 150 is used to tell the program not to generate a new set of values for the problem if the user has typed in a wrong answer to a question. If the answer was wrong, it is presented to the user over and over again until the correct answer is typed in.

The check to determine whether it is time to print out the score is made by looping back to line 100 after each answer. Therefore, if line 150 were not in the program, this looping would cause the program to generate a new problem for presentation to the user, even if the answer to the previous question was wrong. Line 150 ensures that the same problem will be presented over and over until answered correctly.

Lines 210 and 220 generate the random number used for the problem presented to the user. Line 210 as it is written generates a random number between 1 and 10. Line 220 generates a random-number number between 0 and 9. With this combination of numbers the problems presented to the user range between 1 X 0 through 10 X 9. I have tried making changes to these two lines and have had good results using other arrangements. One simple change that could be made if the user was having difficulty with one particular table is to change line 220 from a random number to the table causing the problem. For instance, if difficulty were encountered with the 9s table, line 220 could be changed to read LET B=9, and the program would stay in the table of 9. The questions asked would range from 1 X 9 through 10 X 9.

Line 310 prints the actual problem presented to the user. This line first prints the value the machine has generated for A. Next it prints an X to indicate to the user multiplied by, next the value for B is printed, and finally it prints = . Note that there is a semicolon as the last entry of this line. This causes the request for the answer to be printed on the same line as the question.

Line 320 is the request for user to type in an answer. The answer uses the arbitrary value of C, thus the value of C is typed in by the user as the answer to the question asked. The answer is typed on the same line as the question; this was done as a paper conservation measure.

Line 410 checks to determine whether the value of C typed in by the user is the correct answer for the question presented. This check is made by multiplying the value of A times the value of B and comparing this value to the value of C. If they are equal, the answer is correct and the program loops to the 'correct answer' part of the program (which we will discuss in a moment). If the answer is wrong, the program continues on to the next sequence 510.

If the answer typed in by the user is incorrect, line 510 causes the program to print 'YOU GOOFED . . . TRY AGAIN.' Line 520 scores the error register with one error count. Line 530 then loops the program back to line 100 to find out if it is time to print out the current score.

If the answer given by the user is correct, as stated above the program loops to the correct answer part. This begins at line 600. Line 610 scores 1 in the 'G' (Good) scoring register. Next line 620 generates a random number between 1 and 5. The random number generated is assigned an arbitrary variable of R. This value is used to select one of five different statements to print as a commendation to the user for a correct answer. Lines 625 through 645 use the value of R to select the actual statements to print out. Since R is generated at random any one of the five statements could be printed after any answer.

Lines 650, 660, 670, 680 and 690 are the actual statements for printing. Notice that each of the statements is the same length and each is followed by a comma. This, again, was done to conserve paper and to print a neat looking page for the user. Lines 655, 665, 675, 685 and 695 cause the program to loop back to line 100 after printing out the message selected. The loop back causes the program to check to see if it is time to print out the current score for the user.

When the program determines that it is time to print out the score, the program loops to line 700 which causes the program to print YOUR SCORE NOW IS followed by the value of G (the number correct). This is followed by the words CORRECT AND and the value if any for E. Finally the word ERROR(S) is printed on the same line. Score printing occurs after every 10 answers.

Line 720 is used to terminate the run of the program automatically after 50 questions have been answered by the user. This happens automatically because of line 720. The user is given no control after 50 questions as the limit of the scoring registers is 50 maximum. If

Reprinted with permission from Issue 2 of *Kilobaud*, February 1977

after printing the score, the total number of questions is less than 50, the program will continue with line 730 where the user is asked if the program is to continue.

Line 910 is used to determine whether the last question answered before the score was printed was answered correctly. Line 910 is only reached if there have been fewer than 50 questions asked, and the user has indicated the desire to continue on to the next round of 10. If the previous answer was correct, the program loops to sequence 200 where a new problem is generated. If the last answer prior to scoring was wrong the program goes to line 920. This line causes the printing of the statement, I DIDN'T FORGET YOU MISSED THIS ONE. Note, once again, the printing is followed by a comma so that the problem will print immediately after the statement. Next line 930 loops the program back to print out the same problem that was missed before the scoring. The problem will be the same one because the values for A and B are not changed and the loop returns to a point which is after the random number generation. In the third and fourth rounds of Fig. 2 I tried to show another as yet unmentioned feature of the program. Since the overall program is under the control of Tiny BASIC it is possible to use the arithmetic functions contained therein. This feature can make the program challenging to anyone who likes to play with figures. The idea is to think of a number which when added to a second number, will equal the correct answer to the question presented. One can use division, multiplication, addition, and subtraction. The formulas can be as complex as your version of BASIC will allow. The complexity does not matter so long as the end result is the correct answer to the question asked.

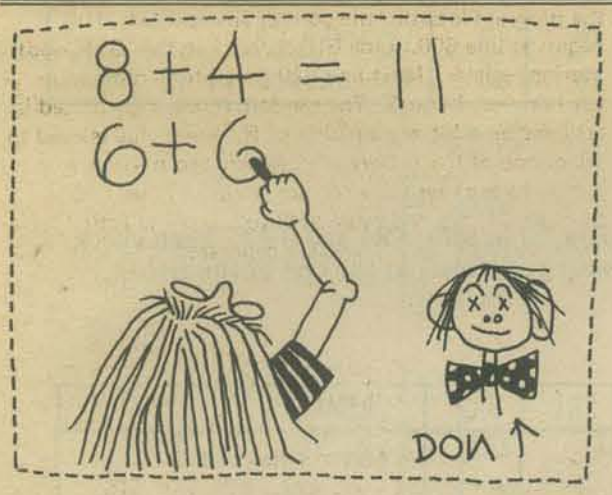
So if your wife is on your back to get that little box to do something and your children need help with their times tables, or if you just want to have some fun here is a simple yet challenging program for you. Happy computing!

PROGRAM LISTING

```
10 REM MULTIPLICATION DRILL PROGRAM,
11 PRINT"MULTIPLICATION PROGRAM . . . DO YOU WANT THE RULES?";
12 INPUT Z
13 IF Z=N GO TO 50
20 PRINT"I WILL PRINT A NUMBER TIMES A NUMBER FOLLOWED BY"
30 PRINT"A QUESTION MARK . . ? . . YOU TYPE IN THE CORRECT ANSWER"
40 PRINT"AFTER YOUR ANSWER TYPE IN A RETURN . . I WILL THEN"
45 PRINT"TELL YOU IF YOUR ANSWER IS CORRECT OR NOT . . ."
46 PRINT"AFTER 10 QUESTIONS I WILL TELL YOU YOUR SCORE . ."
50 REM CLEAR SCORING REGISTERS . . .
60 LET G=0
70 LET E=0
100 IF G+E=10 GO TO 700
110 IF G+E=20 GO TO 700
120 IF G+E=30 GO TO 700
130 IF G+E=40 GO TO 700
140 IF G+E=50 GO TO 700
150 IF C<> A*B GO TO 300
200 REM GENERATE RANDOM A AND B
210 LET A=RND (10)+1
220 LET B=RND (10)
300 REM PRINT PROBLEM AND REQUEST ANSWER
310 PRA,"X";B,"=";
320 INPUT C
400 REM IS ANSWER CORRECT?
410 IF C=A*B GO TO 600
500 REM ANSWER IS WRONG
510 PR"YOU GOOFED . . TRY AGAIN.",
520 LET E=E+1
530 GO TO 100
600 REM ANSWER IS CORRECT . . PRINT RAND, COMMENDATION
610 LET G=G+1
620 LET R=RND (5)+1
625 IF R=1 GO TO 650
630 IF R=2 GO TO 660
635 IF R=3 GO TO 670
640 IF R=4 GO TO 680
645 IF R=5 GO TO 690
650 PR"RIGHT ON . . . . .",
655 GO TO 100
670 PR"GOOD WORK . . . . .",
675 GO TO 100
680 PR"KEEP IT UP . . .",
685 GO TO 100
690 PR"EXCELLENT . . . .",
695 GO TO 100
700 REM OUTPUT SCORE
710 PR"YOUR SCORE NOW IS";G;"CORRECT AND ";E;"ERROR(S)"
720 IF G+E=50 GO TO 800
730 PR"DO YOU WANT TO CONTINUE. .";
740 INPUT Z
750 IF Z=Y GO TO 900
800 PR"THANK YOU FOR PLAYING WITH ME . . HAVE A NICE DAY . ."
810 PR"IF YOU WANT TO PLAY AGAIN TYPE . . RUN . .& . . RETURN . ."
899 END
900 REM PLAYER WANTS TO CONTINUE LESS THAN 50 QUESTIONS
910 IF C=A*B GO TO 200
920 PR"I DIDN'T FORGET YOU MISSED THIS ONE . .",
930 GO TO 300
999 END
```

PROGRAM RUN EXAMPLE

```
RUN
MULTIPLICATION PROGRAM . . . DO YOU WANT THE RULES ? Y
I WILL PRINT A NUMBER TIMES A NUMBER FOLLOWED BY
A QUESTION MARK . . ? . . YOU TYPE IN THE CORRECT ANSWER
AFTER YOUR ANSWER TYPE IN A RETURN . . I WILL THEN
TELL YOU IF YOUR ANSWER IS CORRECT OR NOT . . .
AFTER 10 QUESTIONS I WILL TELL YOU YOUR SCORE . .
2X0=? 0

YOU GOT IT . . .        3X7=? 21
EXCELLENT . . .         4X6=? 24
KEEP IT UP . . .        7X3=? 21
RIGHT ON . . .          6X4=? 24
KEEP IT UP . . .        9X7=? 49
YOU GOOFED . . TRY AGAIN. 9X7=? 63
GOOD WORK . . .         8X6=? 48
YOU GOT IT . . .        1X3=? 3
EXCELLENT . . .         4X2=? 8
KEEP IT UP . . .
YOUR SCORE NOW IS 9 CORRECT AND 1 ERROR(S)
DO YOU WANT TO CONTINUE . . ? Y
3X7=? 21
RIGHT ON . . .          2X4=? 8
YOU GOT IT . . .        7X3=? 21
EXCELLENT . . .         6X2=? 12
KEEP IT UP . . .        1X5=? 10
YOU GOOFED . . TRY AGAIN. 1X5=? 5
RIGHT ON . . .          8X8=? 64
RIGHT ON . . .          1X3=? 3
GOOD WORK . . .         6X8=? 48
EXCELLENT . . .         5X5=? 10
YOU GOOFED . . TRY AGAIN.
YOUR SCORE NOW IS 17 CORRECT AND 3 ERROR(S)
DO YOU WANT TO CONTINUE . . ? Y
I DIDN'T FORGET YOU MISSED THIS ONE . . 5X5=? 25
EXCELLENT . . .         2X2=? 8 – 4
GOOD WORK . . .         3X5=? 30/2
RIGHT ON . . .          8X2=? 64/4
YOU GOT IT . . .        3X7=? 10+4
YOU GOOFED . . . TRY AGAIN. 3X7=? 20+1
EXCELLENT . . .           8X8=? 128/2
EXCELLENT . . .           7X5=? 40–5
KEEP IT UP . . .          10X2=? 22–2
KEEP IT UP . . .          9X9=? 80
YOU GOOFED . . . TRY AGAIN.
YOUR SCORE NOW IS 25 CORRECT AND 5 ERROR(S)
DO YOU WANT TO CONTINUE . . ? Y
I DIDN'T FORGET YOU MISSED THIS ONE . . 9X9=? 81
YOU GOT IT . . .          4X2=? 10–2
EXCELLENT . . .           3X3=? 10–4
YOU GOOFED . . . TRY AGAIN. 3X3=? 10–1
YOU GOT IT . . .          4X0=? 100/10–10
RIGHT ON . . .            5X9=? 90/2
RIGHT ON . . .            10X0=? 200/20–10
RIGHT ON . . .            1X7=? 4+3
RIGHT ON . . .            4X2=? 10–2
GOOD WORK . . .           3X5=? 20–5
YOU GOT IT . . .
YOUR SCORE NOW IS 34 CORRECT AND 6 ERROR(S)
DO YOU WANT TO CONTINUE . . ? N
THANK YOU FOR PLAYING WITH ME . . HAVE A NICE DAY . .
IF YOU WANT TO PLAY AGAIN TYPE . . RUN . . & . . RETURN . .
```

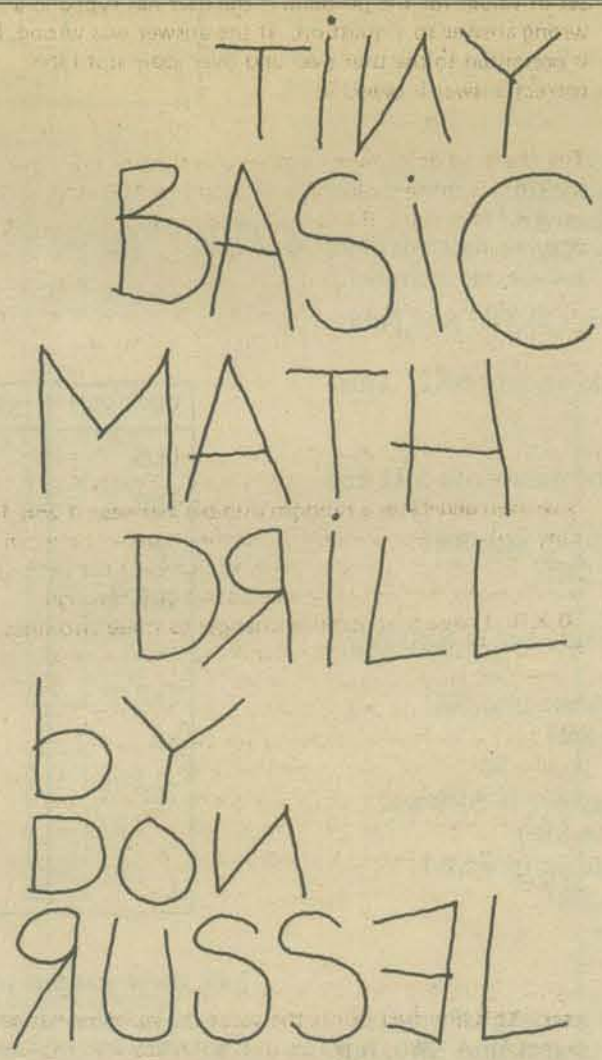


I am enclosing a copy of a drill program for mathematics. It is presently being run on a SWTP 6800 system with TVT-II terminal. This program and Pittman's Tiny BASIC together fit in 4K.

The program first prints a 'menu', from which the operator selects a drill. After 25 problems have been done, a grade is printed. In case the operator gives a wrong answer, the program gives the correct answer.

Lines 340 and 350 may be deleted if a hard-copy device is used; I put them in so the operator could read his score before the program re-started. Control codes were used in PRINT lines 10, 30, 40, 50, and 60 to cause a 'Home-Erase' before printing. Also, I inserted statements to cause 'Home-Erase' after each 10 problems, to prevent overflow in the CRT memory.

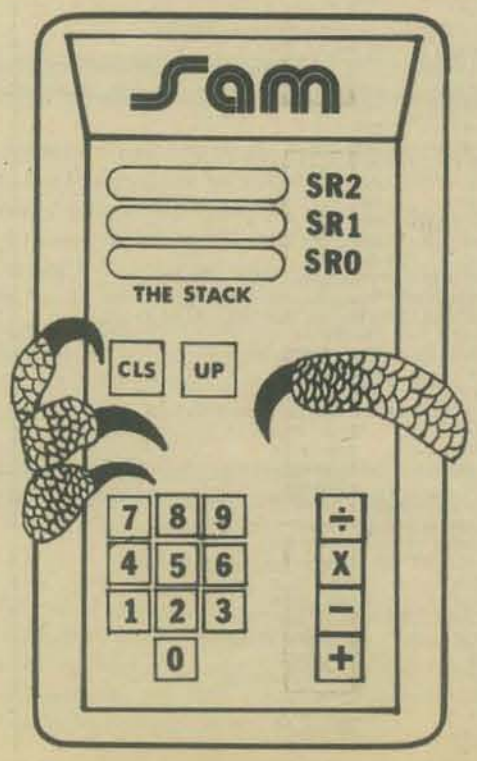
I really enjoy using Tiny BASIC. Keep up the good work.




```
10    PRINT "MATHDRILL"
11    PR
12    PRINT "PUSH 1 FOR ADDITION"
13    PRINT "PUSH 2 FOR SUBTRACTION"
14    PRINT "PUSH 3 FOR MULTIPLICATION"
15    PRINT "PUSH 4 FOR DIVISION"
16    INPUT Z
20    IF Z < 1 THEN GOTO 999
30    IF Z = 1 PRINT "ADDITION DRILL"
40    IF Z = 2 PRINT "SUBTRACTION DRILL"
50    IF Z = 3 PRINT "MULTIPLICATION DRILL"
60    IF Z = 4 PRINT "DIVISION DRILL"
65    IF Z > 4 THEN GOTO 10
70    PR
80    E=0
90    G=0
100   A=RND(8)+2
110   B=RND(9)+1
120   G=G+1
130   IF Z=1 PRINT "  ";A;"+";B;"=";
140   IF Z=2 PRINT "  ";A+B;"–";A;"=";
150   IF Z=3 PRINT "  ";A;" X ";B;"=";
160   IF Z=4 PRINT "  ";A*B;"/";A;"="
170   INPUT C
180   IF Z=1 THEN B=A+B
190   IF Z=3 THEN B=A*B
200   IF B=C THEN GOTO 300
220   E=E+1
230   PRINT "THE ANSWER IS ";B
300   IF G < 25 THEN GOTO 100
310   PRINT "YOUR SCORE IS ";(25–E)*4
320   IF E < 2 PRINT "THAT'S GREAT"
330   IF E >5 PRINT "YOU NEED MORE PRACTICE."
340   PRINT " PUSH 1 TO GO AGAIN"
350   INPUT G
360   GOTO 10
999   END
```

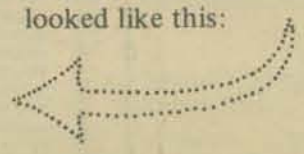
MAKE-BELIEVE COMPUTERS, PART TWO

OR . . . THE FURTHER ADVENTURES OF SAM BY THE DRAGON



In our Nov-Dec issue we introduced SAM, a handheld, make-believe computer. SAM has a *stack* of three *registers*. Each register can hold one number, which must be an *integer*. SAM also has a keyboard for entering numbers into the stack, moving them around in the stack, and performing arithmetic operations.

In our Nov-Dec issue, SAM looked like this:



In case you have misplaced the Nov-Dec issue of *PCC*, here is a brief rundown on how SAM behaves when you press his keys.

[CLS] Clears the stack to 0. Puts zero in each of the stack registers, SR0, SR1 and SR2.

[0] [1] . . . [9] Use these keys to enter numbers into SR0. You *cannot* key numbers directly in SR1 or SR2.

[UP] Pushes numbers in the stack up one place.
(1) Moves the number that was in SR1 into SR2,
(2) Copies the number that was in SR0 into SR1.
This number is now in both SR0 and SR1.

[+] (1) Adds the number in SR0 to the number in SR1 and puts the sum in SR0.
(2) Copies the number that was in SR2 down into SR1. This number is now in both SR2 and SR1.

[−] (1) Subtracts the number in SR0 from the number in SR1 and puts the difference in SR0.
(2) Copies the number that was in SR2 down into SR1.
This number is now in both SR2 and SR1.

[X] (1) Multiplies the number in SR1 by the number in SR0 and puts the product in SR0.
(2) Copies the number that was in SR2 down into SR1.
This number is now in both SR2 and SR1.

[÷] (1) Divides the number in SR1 by the number in SR0 and puts the integer part of the quotient in SR0.
(2) Copies the number that was in SR2 down into SR1.
This number is now in both SR2 and SR1.

And, do you remember?

If we key in a number immediately after pressing UP, the stack is *not* pushed. The number keyed in goes into SR0, but SR1 and SR2 are not changed.

If we key in a number immediately after pressing +, −, X or ÷ , the stack is pushed. The number that was in SR1 is pushed into SR2 and the number that was in SR0 is pushed into SR1.

"INP a" means "key in the numerical value of a," and the number a may have more than one digit

Last time, we showed you some simple *programs*, then left you with some homework to try. Here are *our* answers to some of the homework problems

(1) ab + cd = ?

PROGRAM	SR0	SR1	SR2	REMARKS
CLS	0	0	0	Clear the stack
INP a	a	0	0	Key in the value of a
UP	a	a	0	
INP b	b	a	0	Key in the value of b
X	ab	0	0	Compute ab
INP c	c	ab	0	Key in the value of c
UP	c	c	ab	
INP d	d	c	ab	Key in the value of d
X	cd	ab	ab	Compute cd
+	ab+cd	ab	ab	The answer is in SR0

(2) (a + b)(c + d) = ? *You* fill in SR0, SR1 and SR2. Then check with our answer at the end of the article.

PROGRAM	SR0	SR1	SR2	REMARKS
CLS				Clear the stack
INP a				Key in the value of a
UP				
INP b				Key in the value of b
+				Compute a+b
INP c				Key in the value of c
UP				
INP d				Key in the value of d
+				Compute c+d
X				Answer is in SR0

Yes, there was a typographical error in problem (3). It should have been

(3) $ax^2 - by^2$ = ? Complete the SR0, SR1 and SR2 columns. Answers are at the end of the article.

PROGRAM	SR0	SR1	SR2	REMARKS
CLS	0	0	0	Clear the stack
INP x	x	0	0	Key in value of x
UP	x	x	0	
X	x^2	0	0	Compute x^2
INP a	a	x^2	0	Key in value of a
X	ax^2	0	0	Compute ax^2
INP y				Key in value of y
UP				
X				Compute y^2
INP b				Key in value of b
X				Compute by^2
-				The answer! In SR0

We will skip (4). Do you see how it can be done by starting with the solution for problem (1) and adding a few steps?

(5) $x^3 + y^3 = ?$

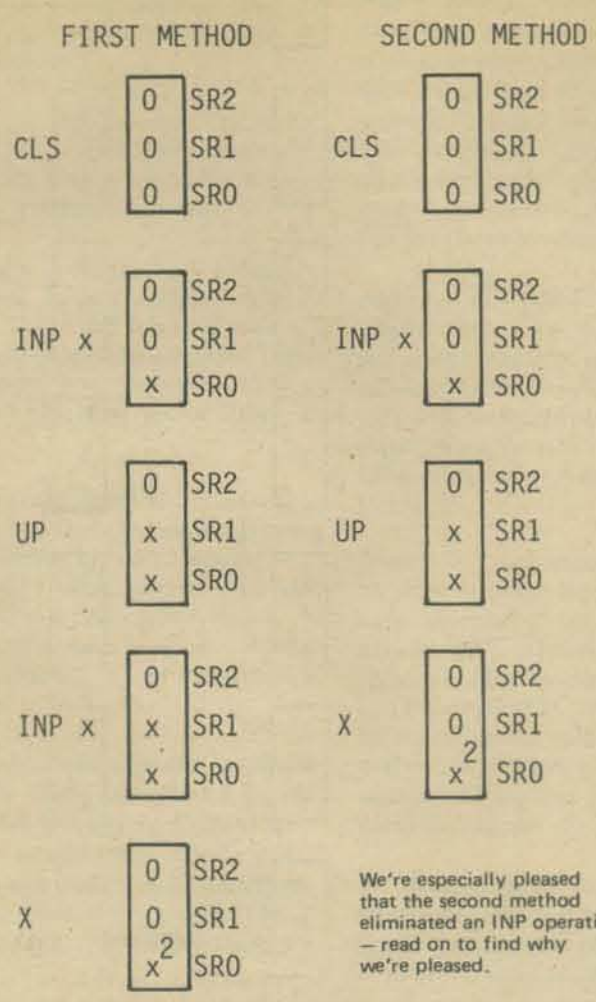
PROGRAM	SR0	SR1	SR2	REMARKS
CLS	0	0	0	Clear the stack
INP x	x	0	0	Key in the value of x
UP	x	x	0	
UP	x	x	x	
X	x^2	x	x	Compute x^2
X	x^3	x	x	Compute x^3

Carry on! We are confident that you can complete the program without any further help. And! We are still waiting for *your* solutions to problems (6) and (7). Please hurry!!!

TRICKS OF THE TRADE

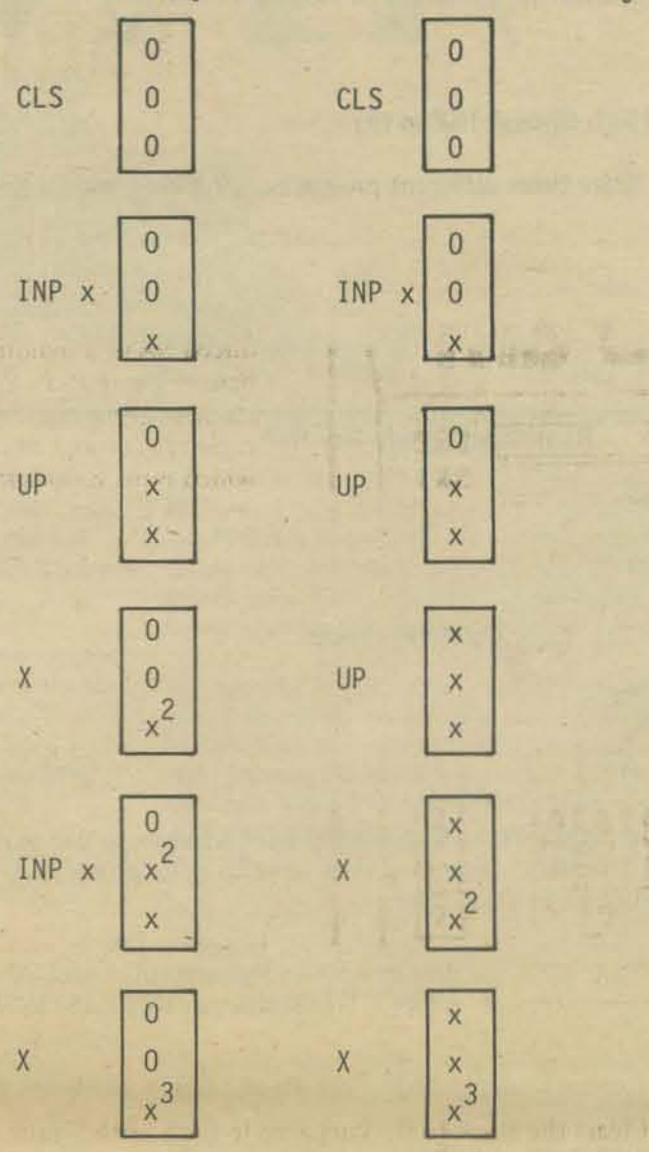
Follow along as we show you some "tricks of the trade," ways to save steps.

(6) $x^2 =$



Watch this one very carefully!

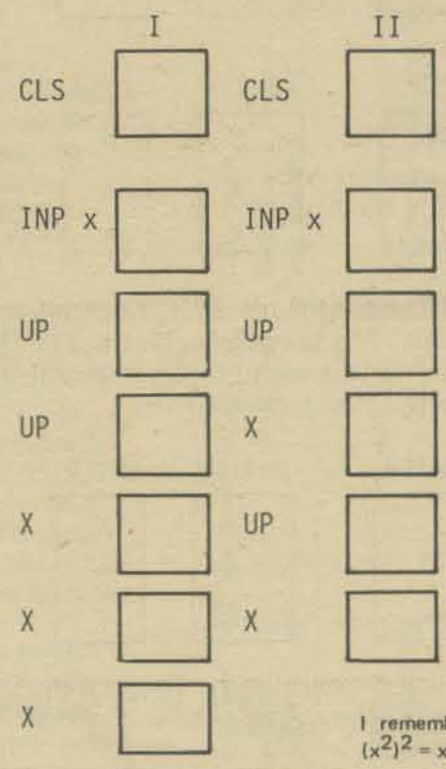
(7) $x^3 =$



We prefer the method with only one INP x, since that reduces chances of error (and it's quicker) when you're dealing with numbers having more than 1 digit. In later issues we'll discuss other reasons for limiting the number of input statements.

YOUR TURN

(8) Here are two ways to compute x^4.
Complete the stacks. Our answers are at the end of the article.



Still your turn. Write programs to do each of the following computations. In each case, use only *one* INP operation in your program. Try to write two or more *different* programs to solve problem.

(9) x^5

(10) a^5 Hint: Change INP in (9)

(11) c^6 Write three different programs. $c^6 = c{\cdot}c{\cdot}c{\cdot}c{\cdot}c{\cdot}c = (c^2)^3 = (c^3)^2$

(12) b^7

(13) m^8

(14) x^{64}

(15) $x^2 + x$ Remember! Only *one* INP.

(16) $x - x^2$

(17) $a^3 + 2a$

(18) $2k^2 + 3k$ Only *one* INP, please.

(19) $s^4 + s^2$

(20) $(r^2 - r)^2$

Try these on SAM. You will probably have trouble, so use more than one INP, if necessary. You may even have to write down partial answers and re-enter them later.

(21) $x^2 - x$

(22) $x^3 - x^2$

Did you find solutions to 21 and 22 awkward? DON'T DESPAIR . . . HELP IS ON THE WAY!

Here are two new keys for SAM. Use them to help you solve problems (21) and (22)

CHS Change the algebraic sign of the number in SR0. This operation will change a positive number to a negative number, or a negative number to a positive number, but zero is not affected . . . it remains zero.

BEFORE	AFTER
5	5
3	3
7	-7

BEFORE	AFTER
157	157
22	22
-3	3

BEFORE	AFTER
8	8
-2	-2
0	0

BEFORE	AFTER
b	b
a	a
x	-x

XCH Exchange the numbers in SR0 and SR1. The number that was in SR0 is put into SR1 and the number that was in SR1 is put into SR0. The number that was in SR2 just sits there quietly, completely unaffected by this operation.

BEFORE	AFTER
0	0
-2	7
7	-2

BEFORE	AFTER
x	x
x	x^2
x^2	x

Do . . . Do . . . Do . . . send your answers, problems, ideas and make believe computers to the PCC Dragon.

ANSWERS

Here are answers to problems (2) and (3), with the stack filled in. We show them without the REMARKS column.

(2) $(a + b)(c + d) = ?$

PROGRAM	SR0	SR1	SR2
CLS	0	0	0
INP a	a	0	0
UP	a	a	0
INP b	b	a	0
+	a+b	0	0
INP c	c	a+b	0
UP	c	c	a+b
INP d	d	c	a+b
+	c+d	a+b	a+b
X	(a+b)(c+d)	a+b	a+b

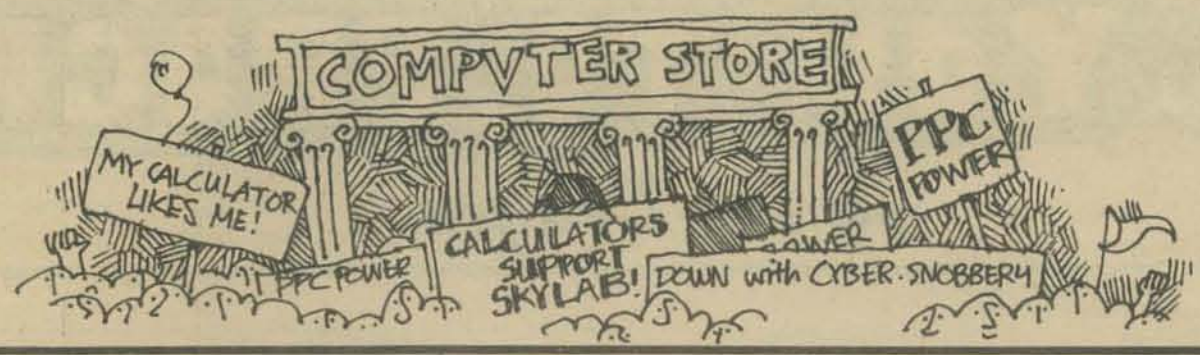
(3) $ax^2 - by^2 = ?$

PROGRAM	SR0	SR1	SR2
CLS	0	0	0
INP x	x	0	0
UP	x	x	0
X	x^2	0	0
INP a	a	x^2	0
X	ax^2	0	0
INP y	y	ax^2	0
UP	y	y	ax^2
X	y^2	ax^2	ax^2
INP b	b	y^2	ax^2
X	by^2	ax^2	ax^2
-	ax^2-by^2	ax^2	ax^2

(8) Compute x^4

	I		II
CLS	0 0 0	CLS	0 0 0
INP x	0 0 x	INP x	0 0 x
UP	0 x x	UP	0 x x
UP	x x x	X	0 0 x^2
X	x x x^2	UP	0 x^2 x^2
X	x x x^3	X	0 0 x^4
X	x x x^4		

·PPC POWER·



by Craig Pearce

Originally published as *The Programmable Pocket Calculator Owner: Who Does He Think He Is?* in The CACHE Register, Vol 1:7; reprinted with permission.

It began innocently enough at the last meeting of CACHE, and the incident has left such a scar on me that I feel I must bring the entire thing out into the open. Of course, the whole thing should have been clear to me from the start, and would have been were it not for my naive faith in human nature.

I was speaking with Ted Nelson, author of the book *Computer Lib and Dream Machines* about the coming age of a computer in every home. It is a good and exciting dream and Mr. Nelson's enthusiasm on bringing the uPs out of the closet and away from the 'cybercrud' types that can still be found veiling these computing devices in shrouds of mystery can not be out done

Since my interests are many, I naturally brought up the subject of the programmable calculators. You'll never guess what Ted – a computer for everyone – Nelson did. He laughed! He flatly stated that if it couldn't do graphics on a CRT it wasn't a computer, and laughed. When he learned that some machines had the ability to store and play back programs on magnetic cards, he roared even more.

"Why bother . . . what for?" he added, equating that feature as being as ridiculous as a somewhat off-color joke in Mel Brooks' "Silent Movie."

Needless to say, I was stunned. Here is a man that wants to see the computer come out from the false complexities that surround it, and make that power available to everyone, and then makes a statement as he did.

Then, slowly, the pieces began to fall in place. I began to see an ever clearing picture. It isn't just Mr. Nelson alone, it's nearly everyone. Didn't Bill Precht himself include the PPC in any survey only when reminded that they exist? Hadn't all attempts to stir interest in a calculator sub-group failed? It's all a clear case of Cyber-*snobbery*.

Ah, you may say, but my machine *is* better. I can control several input and output ports, I can run things in my home, my memory is expandable, my speed quicker. And, granted, it's all true, and I would hardly be the one to want and say that the PPC is actually better than a SWTP 6800, or Altair or Imsai or what have you. After all, I own a uP too. The question is, is a uP better than a PPC? I let you be the judge.

Over the past several meetings, I have seen and heard talk on several pieces of software. These include such interesting ones as diagnostics (to see if the damn machine is working), an 8080-Educator program that actually lets you see 4 whole registers as you input a limited number of commands, one at a time. Then there is a whole list of programs that can transfer data. Now that's really something.

"What can you do with your computer?" a friend asks. "I can run a program that relocates itself in RAM!" you proudly answer. Terrific. Or how about one that will fill a CRT screen with a character or some oddball pattern. Great way to spend an evening.

And what about all those "things" that can be controlled around the house? Heating, air-conditioning and the like. We've all got that programmed in, right? We don't? Takes too long to rewire the house, you say. Don't really want to trust your machine at running all times? I see.

Okay, so what I'm getting at is that while the uP has great promise, all the predictions haven't yet come to pass.

What can the PPC do, however? Probably nothing that you couldn't program your micro to do, or course. Certainly not in the number of steps, however. For example, with the HP-65, the user has 100 6-bit program steps at his disposal. How many can program their micro to multiply two 18 digit numbers and produce a 36 digit product in 100 bytes? How about Hexpawn, or a cybernetic Nimb game? If I were so inclined, I could pass a card through my 65 and load in a 100 step program to perform "Parallax Transformations in a Celestial Reference System". I can also balance my check book, perform trend line analysis, compute components for a chebyshev filter, check male pulmonary functions, navigate a ship, fly a plane by one or two VORs, have a game of Hangman using an alphabetic over-lay of the keys, or simulate a dime slot machine that duplicates all standard payoff combinations. And I can do all this at the time I need it. My machine fits in a pocket and operates from batteries.

An owner of an SR-52 has the ability to do binary searches; linked list; manipulation of subscripted variables and arrays; interrupt processing; dynamic code modification; op code translation; linked editing, loading and execution; overlays paging; and, yes Ted, even output graphics, via the attachable printer. The new HP67/97 series opens up even more advanced programming techniques.

Speed? The programmable calculator is slow. Remember, however, that it runs in an interpretive mode. A loop that takes 15 minutes on a KIM I might take 30 days on a PPC. However, whatever reason for the loop, chances are the function is already available at the touch of a key on the PPC. Accuracy can't be beaten. The PPCs I'm familiar with have 10 digits of accuracy with a range of 1x10**-99 to 9.999999999x10 ** 99 for both positive and negative numbers. That range actually exceeds the volumn of the known universe in cubic microns!

Mr. Nelson predicts that over 10,000 people are going to attend the upcoming convention and that this will really get the public aware of computers. Well, just as a point of interest, many people I've talked to have become interested in this, the greatest of all hobbies, through the PPCs. And there are 70,000 of us. I personally belong to a PPC club from California that has over 1250 members nation wide, with membership growing through word of mouth only. Fact is, the PPC has always had more public exposure than the average micro.

Just who *does* a PPC owner think he is? He (or she) is a person that needs computing power, without the time to wait for time sharing; a person who needs this power at odd times and places that won't allow for some remote terminal. A PPC owner takes pride in accomplishing difficult computing tasks on a small, limited memory machine.

(Did anyone know that two HP-65s went along on the American/Russian skylab mission and were used to back up and confirm the results obtained by the onboard computer?)

Maybe this whole thing has been stated pretty strong. And maybe it has to be. I'm not saying down with the micro-computer. I'm saying down with cyber-snobbery. Maybe the lowly PPC can't ever hope to do all the advanced functions of a genuine micro, and it really shouldn't. But to just laugh, and think that it will never play a role in personal computing is absurd. It already has.

DATA HANDLER: ANSWERS

Here are typical answers to the homework problem on page 18. Your choice of words may be different.

Address	Data	Mnemonic	Mode	Description
FC00	A9	LDA	IMMEDIATE	Load the accumulator
FC01	36			with the HEX number 36
FC02	8D	STA	ABSOLUTE	Store the number in the
FC03	05			accumulator into memory
FC04	FD			location FD05
FC05	4C	JMP	ABSOLUTE	Change the program
FC06	05			counter to FC05.
FC07	FC			Continually loop through steps FC05, FC06 and FC07 until halted.

GAMES FOR YOU TO PROGRAM

by the Dragon

REVERSE for a Video Terminal

Last time, you may recall, we described nine different variations of the game REVERSE. For you lucky people who have a video terminal with an addressable cursor, here is a version of REVERSE just for you. This version features some simple "animation" on the screen as the numbers get reversed right before your eyes. Ready? Here we go . . . the game starts like this.

```
RUN

THIS IS THE GAME OF REVERSE.  I WILL GIVE YOU A
SCRAMBLED LIST OF NUMBERS.  YOU UNSCRAMBLE THE LIST ...
PUT IT INTO NUMERICAL ORDER.  FOR EXAMPLE,

SCRAMBLED LIST:       2 5 1 4 7 6 3

PUT IT IN THIS ORDER: 1 2 3 4 5 6 7

WHEN YOU ARE READY, PRESS RETURN AND I WILL TELL YOU MORE
ABOUT THE GAME OF REVERSE
```

CURSOR

Easy enough so far . . . we press RETURN and:

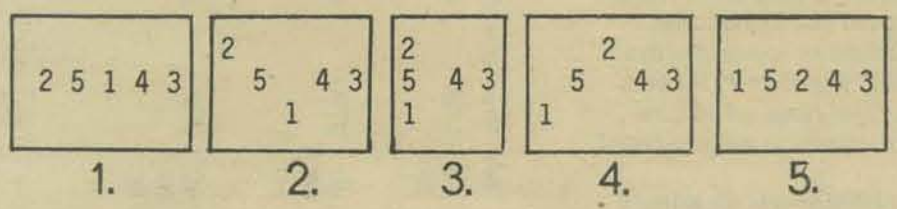
```
YOU MAY REVERSE TWO OR MORE NUMBERS IN THE LIST, STARTING
FROM THE LEFT END OF THE LIST.  FOR EXAMPLE,

  ORIGINAL LIST:   2 5 1 4 7 6 3

  WE REVERSE 3:    1 5 2 4 7 6 3

YOU MAY CHOOSE THE SIZE OF THE LIST.  YOU MAY ASK FOR UP
TO 9 NUMBERS.  HOW MANY NUMBERS DO YOU WANT?
```

Let's try an easy game. We type 5 and press RETURN.

```
SCRAMBLED LIST:  2 5 1 4 3  HOW MANY SHALL I REVERSE?
```

Obviously (we think), we should REVERSE 3 so that 1 will be in the first place. So, we type '3' and press RETURN.

Wow! If we could just show you the action on the screen in glorious SUPER–8! Since we can't, here is a frame-by-frame description of what happened.



The first three numbers, from the *left* end of the list, have been REVERSED.

ORIGINAL LIST NEW LIST

2 5 1 4 3 1 5 2 4 3

Well, we did get 1 in the first place, but the rest of the list is still scrambled. Here is the current state of affairs.

```
SCRAMBLED LIST:  1 5 2 4 3  HOW MANY SHALL I REVERSE?
```

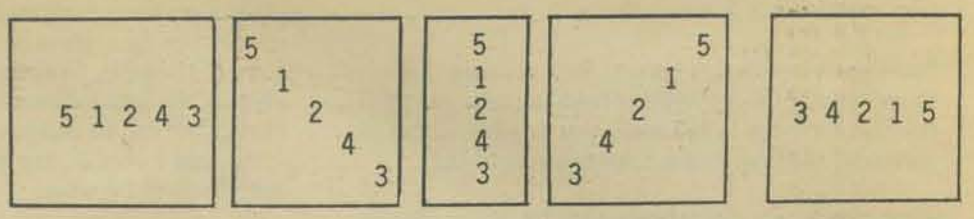
Let's reverse 2. That should put the 1 and 2 together. We type 2 and press RETURN. The 1 and the 5 march around on the screen and settle into their new places.

```
SCRAMBLED LIST:  5 1 2 4 3  HOW MANY SHALL I REVERSE?
```

Aha! 5 is now in the first place. If we REVERSE 5, we will put 5 in the fifth place! So . . . we do it.

```
SCRAMBLED LIST:  3 4 2 1 5  HOW MANY SHALL I REVERSE?
```

Just to remind you, we will show the action during that last move. All five numbers will move on the screen!



When you program this, be sure to leave enough room above and below the list, so that the numbers have room to move in. Remember, there can be up to 9 numbers in the list.

Eventually, we did put the list in order and the computer displayed the following message.

```
UNSCRAMBLED LIST:  1 2 3 4 5  YOU DID IT!!!

DO YOU WANT TO PLAY AGAIN?
```

If you type YES, the computer will begin again, asking you how many numbers you want. If you type NO, the computer might say something like: THANKS FOR PLAYING WITH ME. LET'S PLAY AGAIN SOMETIME.

YOUR TURN. Write the program. If you do, we would sure like to see it. If you send us the program, please tell us what version of BASIC (or other language) you have used and the name of your video terminal.

THE YES–NO GAME

Program by Carl Main

At last! A solution to one of our "Games for *YOU* to Program." Here is a listing and run of the YES–NO GAME which we suggested on page 13 of the Nov–Dec 1976 issue of *PCC*. This program is by Carl Main, Shoreline Community College, 16101 Greenwood Avenue North, Seattle, WA 98133. Carl uses a PRIME 300 computer with a form of BASIC somewhat different from our BASIC. We have made some notations to help you read this program.

Caution! The program does have a couple of bugs. We talked to Carl and he is fixing them. Meanwhile, here are two bugs that we know about: (1) typing a question with leading or imbedded spaces causes problems and (2) the program will fail on questions such as **>9** but will work if you type **>09** instead.

SHALL WE CONTINUE THIS SERIES???

So far, only one person has responded in *any* way to "Games for YOU to Program." If people are not interested in this sort of thing, we will drop the series in favor of something you *do* want – just let us know.

If you do send us a program in response to one of our game suggestions, please also send a RUN of the program. Both LISTing and RUN on nice white paper with crisp black printing, please! Even better, send us paper tapes of both the program and a lengthy RUN of the program.

We also need to know what computer and what version of BASIC (or other language) you used to write the program. If you have the time, it would sure help everyone read the program if you could annotate it, on separate paper from the listings, please.

If you don't feel like writing a program for a game we suggest, how about inventing a game for someone else to program?

```
100 REM- YES-NO GAME
110 REM- CARL MAIN, SHORELINE COMMUNITY COLLEGE, SEATTLE WA 98133
120 PRINT 'I WILL THINK OF A NUMBER FROM 0 TO 99. YOU MAY ASK'
130 PRINT 'ME QUESTIONS LIKE THE ONES SHOWN BELOW.'
140 PRINT '>50 (BUT WITH *YOUR* NUMBER AFTER >)'
150 PRINT '<75 (BUT WITH *YOUR* NUMBER AFTER <)'
160 PRINT '=39 (BUT WITH *YOUR* NUMBER AFTER =)'
170 PRINT
180 PRINT 'WHEN I ASK FOR YOUR QUESTION, YOU MAY ASK ANY ONE OF'
190 PRINT 'THE ABOVE THREE TYPES OF QUESTIONS. THEN I WILL ANSWER'
200 PRINT 'EITHER YES OR NO.'
210 X=INT(100*RND(0))
220 PRINT
230 PRINT 'YOUR QUESTION';
240 INPUT Q$
250 B$=' '
260 B=1
270 S$='<=>'
280 D$='0123456789'
290 P=0
300 FOR I=1 TO 3
310 IF SUB(Q$,B)=SUB(S$,I) THEN 350
320 NEXT I
330 PRINT 'I DO NOT UNDERSTAND. ASK AGAIN.':
340 GOTO 220
350 B=B+1
360 FOR J=1 TO 10
370 IF SUB(Q$,B)=SUB(D$,J) THEN 390
380 NEXT J
390 P=P+1
400 IF P=3 THEN 420
410 N(P)=J-1
420 IF B<3 THEN 350
430 IF P<2 THEN 460
440 N=10*N(1)+N(2)
450 GOTO 500
460 IF P=1 THEN 490
470 PRINT 'I DO NOT UNDERSTAND. ASK AGAIN.':
480 GOTO 220
490 N=N(1)
500 ON I GOTO 560,640,600
510 PRINT
520 PRINT 'ANOTHER NUMBER':
530 INPUT A$
540 IF SUB(A$,1)='Y' THEN 210
550 STOP
560 IF X<N THEN  PRINT 'YES'
570 IF X<N THEN 220
580 PRINT 'NO'
590 GOTO 220
600 IF X>N THEN  PRINT 'YES'
610 IF X>N THEN 220
620 PRINT 'NO'
630 GOTO 220
640 IF X=N THEN  PRINT 'YES! THAT IS IT! MY NUMBER WAS';X
650 IF X=N THEN 510
660 PRINT 'NO'
670 GOTO 220
680 END
```

Note: strings enclosed in *single* quotation marks. Our BASIC uses double quotation marks.

Our BASIC wants comma or semicolon here.

SUB(Q$,B) is the Bth character of Q$. SUB(S$,I) is the Ith character of S$.

I will be 1, 2 or 3. This happened back in lines 300 - 320.

```
>RUN
I WILL THINK OF A NUMBER FROM 0 TO 99. YOU MAY ASK
ME QUESTIONS LIKE THE ONES SHOWN BELOW.
>50 (BUT WITH *YOUR* NUMBER AFTER >)
<75 (BUT WITH *YOUR* NUMBER AFTER <)
=39 (BUT WITH *YOUR* NUMBER AFTER =)

WHEN I ASK FOR YOUR QUESTION, YOU MAY ASK ANY ONE OF
THE ABOVE TYPES OF QUESTIONS. THEN I WILL ANSWER
EITHER YES OR NO.

YOUR QUESTION !>50        So . . . Q$ is now"750"
YES                       <- Our BASIC puts a question mark here.

YOUR QUESTION !>75
NO

YOUR QUESTION !<62
NO

YOUR QUESTION !>68
YES

YOUR QUESTION !72         <- First character must be <, = , or >
I DO NOT UNDERSTAND. ASK AGAIN.

YOUR QUESTION !=72
NO

YOUR QUESTION !=73
YES! THAT IS IT! MY NUMBER WAS 73

ANOTHER NUMBER?  . . . and so on.
```

A REALIZABLE PHANTASIE

by Jim Day

From time to time, various authors have based their plots on the game of chess. For example, Lewis Carroll's *Through the Looking Glass* is loosely structured as a chess game. It might be amusing to experiment with a program that would convert chess games into simple narratives. To do this shouldn't be especially difficult, although doing it with finesse might prove something of a challenge. Maybe a contest could be arranged, with a suitable prize (an IBM FS kit?) for the program producing the best stories.

For the graphically inclined, each game could be illustrated by a computer-generated video display showing the results of each move. This would not be just a plot of a chess board, but rather a series of stylized line drawings, halftones, or full color renderings, with or without animation and sound effects (or vocalization), depicting a scene from the story. The generated text could be as simple or as comprehensive as one might like, ranging in complexity from one-liners like "Sir Percival, King Whitlock's bravest knight, did battle on the Plain of Salisbury and stormed the castle of King Redbeard" to pages of purple prose (or poetry, for those who prefer overkill.)

The game could be held between two people (refereed by the computer), one person and the computer, two computers, or the computer and itself. The story could be output concurrently with the game or as an "instant replay," and it should be possible to reconstruct the game from the narrative.

Any volunteers? I'd try it myself, but my 8080 only has 256 bytes of RAM and I'm not sure I can fit all of those features into that short a program.

[Editor's note: on Los Angeles TV I've seen 10 minute chess game spots narrated as stirring medieval battles – really fine. Anyone know more about this?]

GAMES

Mine 8
by Doug Felt

In response to your urgent, desperate request for games of any sort, I have sent you my most recent game – Mine8, the product of several months of inspirational frenzy. I have wanted to send you a game before, but I decided to wait until Mine8 was at least presentable. (It still has minor adjustments, such as probabilities of random statements, and perhaps a time limit problem, but is challenging anyway.)

There are other projects in the works at the *** Wizzz Council ***, a certain group of dedicated computer game addicts at my high school (Edward W. Clark). We have written (not copied, written) several games, some of which may be familiar – Mastermind, Life, Keno – and some which are entirely original – Mine8 and UFO. Others are in the planning and development stages – Monte Carlo, a large game combining typical gambling games (a natural, coming from Las Vegas), and Saucer, a 'real state' game in which the commander of a flying saucer dodges mountains, phaser defense guns, and an enemy saucer while trying to beam aboard the people held captive in the enemy base. If it is games you want, we've got them – perhaps.

I say perhaps because our programs are not time- or space- efficient. I think nothing of randoming a 65 by 65 array with any of 10 different types of terrain laid out in different patterns, for instance, or having the computer random a 40 by 40 'mountain range'. We are on time-sharing, paid for by the school district, and thus become a bit extravagant. I do hope this is not a major problem for you.

The other reason I say perhaps, well, we are game-makers, not computer buffs. We know next to nothing about the computer, but a lot about games – such is our particular interest in your magazine, and in computing. I know little about different versions of BASIC, but can translate, so to speak, almost at once, though, and I am certain that our BASIC can be readily altered if it does not match exactly with someone else's.

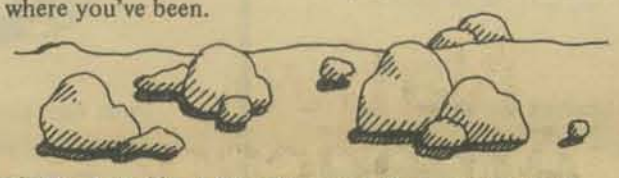
Well, back to Mine8. Perhaps you would like a little more explanation besides that given in the instructions. Here you are –

The aim is to cross the minefield and stay alive, under a time limit of three minutes. You can see only small area around you, sometimes more, sometimes less, depending on where you stand and what has just happened to you. You then enter the direction you wish to turn, and how long you intend to travel, and if the inputs are acceptable, away you go. The maximum amount of time you can travel in one 'turn' is 8.5 seconds, and an input larger than that is refused. An input less than or equal to zero allows you to change your direction again. The actual distance you can travel on clear land is related to your pain level – the higher it is, the slower you go. A pain level of zero allows you to travel three squares (30 ft.) in a little over one second. If it turns out that you haven't moved, you simply lose that time and have to allow more.

Once you are off and running, you will notice that the scene shifts as you turn. You are always faced 'straight,' and all turns are relative – that is, two left turns equal one retreat, four equal a straight – and are in no way related to compass directions. However, at the start you are always facing the correct direction.

There are quite a few types of terrain, each of which has different effects on different things. They are –

1) Clear land. This is what you always wish to travel on, but never can. It lets you see a 7 by 7 area around you, and lets you travel at maximum speed.
2) Hills. Hills slow you down, except for the peak of a hill which always stops you. The great benefit of hills is that they let you see a 9 by 9 area, and 11 by 11 area on the peak of a hill. However, they also expose you to heavier sniper fire.
3) Bushes. More or less the opposite of hills, they restrict your vision to 5 by 5 areas or 3 by 3 when in the center of a bush, and lessen the chance of sniper fire. They also raise your pain level, however, and stop your progress or reduce it to only one additional square.
4) Ledges. The seemingly never-ending monsters which cause three fourths of the trouble. You see, one can never cross a ledge, but must always go around, and ledges can be 25 squares long. Also fun is travelling 20 squares trapped between two ledges, and meeting a mine.
5) Rocks. Annoying, because you can travel a maximum of three squares per turn over them in the first one third of the time. The other two thirds, you trip and raise your pain level (or break your nose.) But who wants to play it safe and only go two squares per turn? Rocks are also fun because you leave no tracks over them, so when you lose your way, you don't know where you've been.



6) The Lake(s). A lake stops you, then on your next turn allows you to start swimming, after you find out how deep it is. Like rocks, you leave no tracks, and although it is not too much harder to swim than it is to run, the lake seems to keep getting deeper, and when it gets too deep, you drown. Nice.
7) What we've all been waiting to see – mines. Mines are really enjoyable, because so much can happen to you. All at once. Wham. Mines aren't always seen, for instance, and they seem to appear suddenly when you are a mere ten squares from the border after only 60 seconds. Surprise! Any mine can do any or all of these things to you:

a) Stun you. Boy, are you lucky, it must have been a dud. But you have been bruised, and this will tell later.
b) Knock you unconcious for up to 15 seconds. When you only have 10 seconds left, naturally.
c) Break one or more limbs. Conceivably, you are expected to crawl along with a maximum of two broken legs, a broken arm, and a broken nose. Of course, if you hit a mine in this state, which would probably be due to a suicidal wish on your part anyway, you will break your neck, and end that suffering.
d) Throw you up to 212 feet away from the mine. Of course, at this distance, you would be dead. Being thrown is the most agonizing thing that can happen to you, because you can become totally lost, without any idea of where to head. The only thing to do is search for your tracks – which is when you remember that you have been struggling over rocks for the last hundred feet, and have left no tracks.
e) And, of course, kill you outright. It does happen.

Hitting a mine almost always reduces your vision to a 3 by 3 area, and spins you around to face in a different direction. Mines are devious, and sometimes you will be congratulated on being alive and then told that you have broken both legs, been knocked unconcious for 13 seconds, been thrown 146 feet, and have only a few short moments to live. Naturally, a mine always increases your pain level, depending on the severity of the injuries inflicted upon you.

A final type of terrain is the border of the minefield, which is filled with bands of the enemy just waiting to get you. You usually cannot escape from them, which is why you never start out a game by going 'retreat.'

If you survive all the pain and suffering, and cross the USA border (a line of exclamation points, usually with a ledge right in front of it), you will be rewarded with a complete map of the minefield, showing your path marked as always with little plus signs. This map is what the game rewards you with – which is why I cheat, knowing the program, and get it printed out even when I lose. To each his own, I suppose. But I never (well, almost never) print it out first, before I enter the minefield, for I do have *some* sense of propriety.

What I almost put into the minefield, but never finished, was the 'obstructed vision' which would have eliminated areas behind a hill or bush from view. If I do complete this, I'll send it along to you.

Mine8 is what I would call a 'level 4' game, on a scale of 0 for extremely simple (pick a number between one and 5) games, and 6 for very complex games, with Star Trek rating about 4.3. Saucer 1 will probably be about 5.1, if it is ever finished, with 'Command,' an imaginary, ultimate game, which I could only dream of actually completing, rating a 6. (A sample command for the flight of the *main* ship in 'Command' –

```
# 8   ? *Energy Vert (500 step 10 Till 600 Time
        4.5 Set 3)(Power 25000 Alt 400
        Continue? Speed 650.3 (Danger 4 Angle
        .02))/ (400'5)(300'5)(Flag 1)
        Continue? (Repeat)(Goto 23)/
```

– which is just a *trifle* complex. Saucer1 is a much reduced version of this game.) My rating scale is almost exponential, as you can see.

Well, enough for one letter. My name and address:
Doug Felt (or the *** Head Wizzzz ***)
2613 Burton Ave.
Las Vegas, NV 89102

* *

Thanks Doug! Sorry for the delay in publishing this: Dragons aren't known for keeping treasures in the best of order, and this gem got hidden. We haven't published the listings because they were so long, and also we need super-high quality stuff with a new ribbon on white paper. (Otherwise reproduction produces illegible stuff, as you've doubtless seen.) The best way around this problem is to send PCC paper tapes, clearly labeled with

- *game name .*
- *your name*
- *size of program*
- *machine coded for*
- *date*

Then we can generate a fresh, unfolded listing for publication. And we can give the tape to CCC who can then very cheaply supply a paper tape of your game to others.

We're thinking about publishing a "PCC Annex" which would be margin-to-margin listings. Longish listings, such as Mine8, could well appear in such a publication.

Simulations
The WORLD of SELLING
by MECC

Learning to BUY, learning to SELL. That is what the Free Enterprise System is all about. One way of teaching kids about being a consumer is to teach about selling. That is the purpose of a series of computer programs called SELL, developed by the Minnesota Educational Computing Consortium (MECC).

Four programs have been written in BASIC to let students experience and learn some of the basic concepts that govern the business world. Written for elementary school kids, SELLAPPLES, SELLPLANTS, SELLLEMONADE and SELLBICYCLES introduce and have students work with concepts such as Price, Advertising, Income, Profit, Expenses, Production levels, Assets and Competition. Students are free to experiment and try different strategies to increase their profits without "experienced" adults protecting them from financial disaster.

The four simulations build on each other with concepts introduced one at a time so students have a feel for the basics before they reach the more complex programs. Already shown successful in elementary classes, Jr. and Sr. High teachers are also finding them valuable tools for their students. Why not give them a try?

SELLAPPLES

Students are given an unlimited supply of apples to sell. Their job is to find the BEST PRICE to charge for their apples. Each day the students set the price for each apple and then are told how many apples they sold and what their income was for the day. By changing their price each day the students analyze how PRICE affect the NUMBER SOLD and INCOME.

SELLPLANTS

This program adds the concept of ADVERTISING to the skills developed by running Sellapples. Students are put in charge of selling tomato plants they have raised. On the first day, they set the PRICE they are going to charge. They then have the option on the remaining days to spend money to make advertising signs to advertise their product. By making more or less signs they investigate the effect that ADVERTISING has on INCOME and PROFIT.

SELLLEMONADE

As a class activity, small groups of students make decisions affecting their Lemonade stand. Each day the stands must decide:

PRODUCTION LEVELS	1) How many glasses to make?
ADVERTISING	2) How many signs to make?
PRICE	3) How much to charge for each glass?

The stands compete to see which group can do the best job of managing their business. Random events such as Thunderstorms, Street Construction, Heat Waves and Inflation add realism and excitement to the simulation.

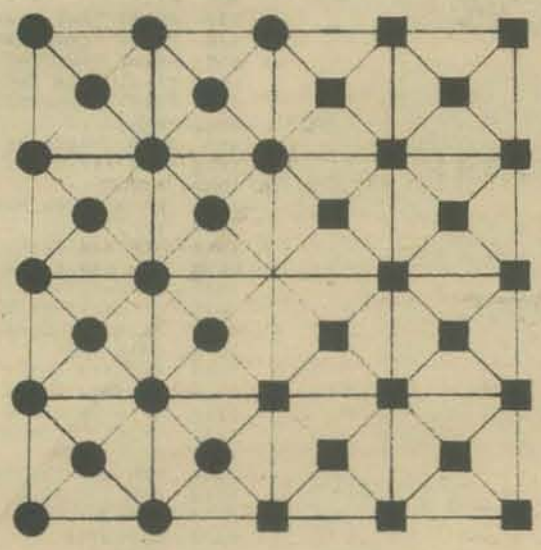
SELLBICYCLES

This program is a direct adaptation of the Huntington II program MARKET. Modifications have been made to make the program easier to use with elementary school students. This program adds the concept of DIRECT COMPETITION between companies. Unlike SELLLEMONADE, the decisions made by one company have a direct effect on the sales of the other company.
A complete manual for the programs in the SELL series with background information, list of objectives, lesson plans, introductory student worksheets, and instructional units can be obtained by writing:

MECC Publications Office
2520 Broadway Drive
Lauderdale, Mn 55113

Native American Board Games

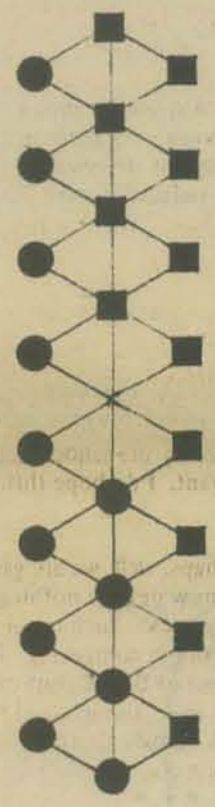
This article is reprinted with permission from **Gamesemag** *Vol. 1 No. 8, "a newletter for people who enjoy games." We'll reprint articles that might be of interest to PCC folk. Who'll be first on the block to have Native American Board Games on a home computer?*



TUKNANAVUHPI is a Hopi chase game played on a 4 x 4 board with diagonals drawn in all of the squares. Pieces move on the points where lines meet on the board and not on the squares. There are two players, each of which has 20 pieces placed on the board as indicated in the drawing. Notice that the point on the board left empty is the one in the middle.

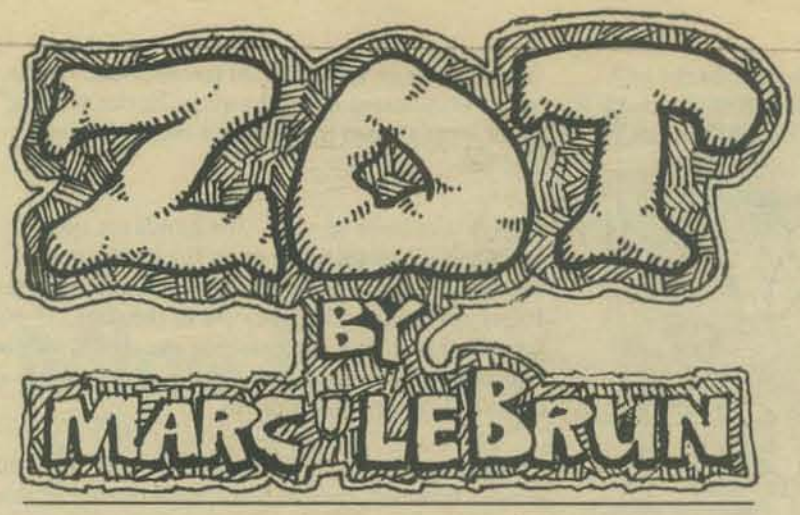
Players take turns moving. The moves are made in any direction along the lines on the board, one line at a time (i.e., from one intersection to another). Pieces are captured like they are in checkers. If you jump over an opponent's piece *in any direction*, you capture it. The winner is the player who captures all of his or her opponent's pieces.

This game is similar to Fox and Geese and other chase games. However, there is one special feature of this game that makes it unique among all the chase games we have seen. When any line across one side of the board is empty of all pieces that line is not used in the game. The board shrinks. As the game goes on the board gets smaller and smaller so that games do not end inconclusively with a few pieces scattered over the board. In developing strategy for playing Tuknanavuhpi one has to think of taking advantage of the changes in the board as well as the position of the pieces.



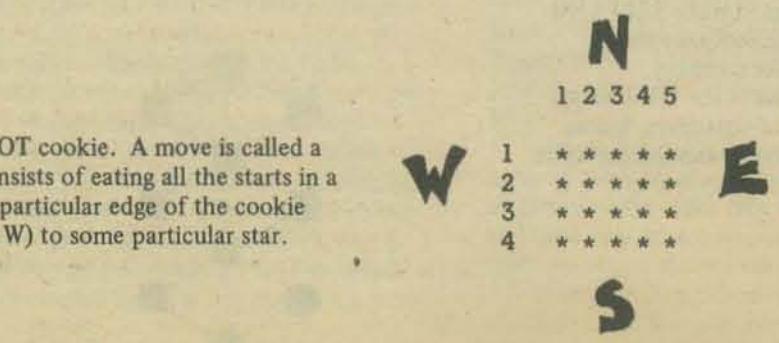
AWITHLAKNANNAI means "stones kill". This Zuni chase game is played on a snake like board with either 25 points or 49 points. Usually the boards are carved out of wood or drawn in the ground, and the pieces consist of 2 different kinds of stone. The stones are placed on the board as indicated by the drawings. Just as in Tuknanavuhpi the point in the middle of the board is left open. Pieces move one step along any line that passes through the point they stand on. They move from point to point and capture by jumping over an opponents piece onto an empty point. The first player to capture all of the opponents pieces wins. It is easy to draw a board on a piece of graph paper and use stones or pennies or small poker chips as pieces.

Gamesemag *is published by*
The Center for Open Learning and Teaching
P.O. Box 9434
Berkeley, CA 94709
Editors are Herb Kohl, Ray Nitta, and Mike Orkin. Sample copies are $.40; a 9-issue subscription costs $4.00.



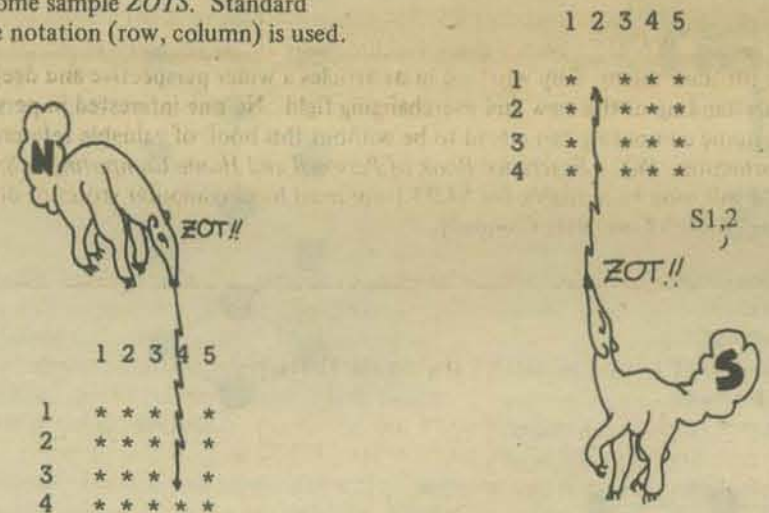
We made this game up while putting together the PCC games book. Since it involved some interesting programming I decided to write it up (it's in HP BASIC) in such a way as to be easy to follow and understand. From the number of REMark statements it should *accompany* a program rather than be incorporated in it. This program is offered in the spirit of providing an example of how to generate programs which are easy to learn from as well as do the intended job.

In ZOT two (human) players compete to get the last bite of a 4X5 rectangular cookie whose sides are labeled with compass directions.

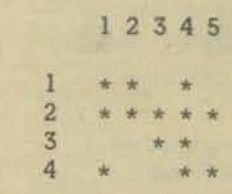


Here is a ZOT cookie. A move is called a *ZOT*: it consists of eating all the starts in a line from a particular edge of the cookie (N, S, E, or W) to some particular star.

Here are some sample *ZOTS*. Standard coordinate notation (row, column) is used.

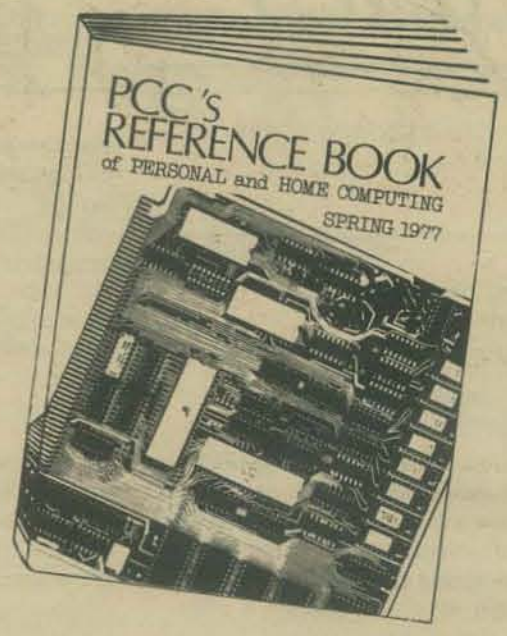


A restriction is that all *ZOTS* must be consecutive. For example, if the cookie looks like the one at the left, then E3,3 and W1,2 are OK, but *not* W1,4. Holes are OK in front of the stars (like E3,3) but not between stars (like W1,4).




```
1  REM *** THE GAME OF ZOT
2  REM *** PEOPLE'S COMPUTER COMPANY : AUGUST 9, 1974 (ABDICATION DAY)
3  REM *** PROGRAMMED BY : MARC LE BRUN
100  REM *** PROGRAM STRUCTURE
110  REM ***
120  REM *** AT THE START OF THE RUN THE BOARD (B) AND STRINGS FOR
130  REM *** THE SIDE TO MOVE FROM (S$) THE NAMES OF THE COMPASS
140  REM *** DIRECTIONS (D$) AND ARBITRARY ANSWERS (A$) ARE
150  REM *** DIMENSIONED AT LINE 700. INSTRUCTIONS ARE PRINTED
160  REM *** BY THE BLOCK OF LINES BEGINNING AT LINE 800.
200  REM *** THE GAME PROPER BEGINS AT LINE 1000 WHERE THE NUMBER OF
210  REM *** ROWS AND COLUMNS (0<R,C<11) ARE INPUT. THE BOARD B IS
220  REM *** FILLED WITH STARS (=1) , THE AUXILLIARY VALUES R1 AND C1
230  REM *** (USED IN VARIOUS CONDITIONALS) ARE INITIALIZED, AS ARE
240  REM *** THE TOTAL NUMBER OF STARS IN THE BOARD (N) AND THE PLAYER
250  REM *** WHOSE TURN IT IS (P).
260  REM *** THE LOOP TRAVERSED FOR EACH MOVE BEGINS AT LINE 2000.
270  REM *** FIRST THE BOARD IS PRINTED. THEN THE SECTION FOR FETCH-
280  REM *** ING THE ACTUAL MOVE IS ENTERED (AT LINE 3000). THE SIDE
290  REM *** OF THE BOARD (S$) IS INPUT, CHECKED, AND CONVERTED TO A
300  REM *** NUMERICAL INDEX (D). THEN THE MOVE ROW & COLUMN (I0,J0)
310  REM *** ARE FETCHED AND CHECKED. A SOMEWHAT COMPLICATED SECTION
320  REM *** FOR CHECKING AND EXECUTING THE ZOT IS BEGUN AT 4000.
330  REM *** THE BASIC PROCEDEURE IS TO BEGIN AT THE SPECIFIED ROW &
340  REM *** COLUMN AND SCAN OUTWARD (WITH CO-ORDINATE INDICIES I & J)
350  REM *** TO THE SIDE OF THE BOARD SPECIFIED BY D. THIS IS ACCOMP-
360  REM *** LISHED BY USING TWO INCREMENTS I1 & J1. ONE OF THESE IS
370  REM *** ZERO AND THE OTHER + OR - 1, DEPENDING ON THE DIRECTION
380  REM *** OF THE ZOT. I1 & J1 ARE DERIVED BY BLACK MAGIC FROM D
390  REM *** VIA THE TEMPORARY VARIABLES D0 AND S, WHICH PLAY NO FUR-
400  REM *** THER ROLE IN THE ALGORITHM.
410  REM *** IN THE INITIAL SCAN A SWITCH (S0) =1 INDICATES
420  REM *** SCANNING OVER STARS, =0 SCANNING OVER SPACE. THE TRANSI-
430  REM *** TION (OF S0) 1->0 IS LEGAL (TO ALLOW ZOTTING OVER LEADING
440  REM *** SPACES). A 0->1 TRANSITION INDICATES THAT THE STARS ARE
450  REM *** NOT CONSECUTIVE. A COUNTER (C0) OF THE NUMBER OF STARS
460  REM *** SCANNED IS KEPT TO #1 SIMPLIFY THEIR REMOVAL IF THE
470  REM *** ZOT IS LEGAL AND #2 TO DETECT A ZOT CONTAINING NO STARS
480  REM *** (WHICH CONDITION IS NOT CAUGHT BY THE S0 'TRANSITIONS')
490  REM *** IF THE MOVE PASSES ALL THESE TESTS THE INITIAL C0 POSI-
500  REM *** TIONS ARE RESCANNED (WITH THE HELP OF A DUMMY INDEX K)
510  REM *** AND SET TO ZERO. THEN N IS UPDATED, IF I1'S ZERO THE LAST
520  REM *** STAR HAS BEEN ZOTTED AND WE ENTER A TERMINAL DIALOG. IF
530  REM *** THE GAME'S STILL AFOOT WE SETUP FOR THE NEXT PLAYER AND
540  REM *** RE-ENTER THE LOOP FROM THE TOP (I.E. LINE 2000).
550  REM ***
560  REM *** AT LINE 5000 A SUBROUTINE THAT INSISTS ON A YES OR NO
570  REM *** REPLY BEGINS. A IS SET TO 1 FOR 'YES', 0 FOR 'NO'.
580  REM ***
590  REM *** THESE REMARKS SHOULD SIMPLIFY UNTANGLING THE CODE.
600  REM *** HOWEVER, BASICK IS HARDLY A TRANSPARENT LANGUAGE.
610  REM *** THIS CODE IS DESIGNED TO BE SIMPLE AND COMPREHENSIBLE,
620  REM *** NOT 'EFFICIENT' (WHATEVER THAT MEANS) !!!
630  REM ***
700  REM *** ONCE ONLY INITIALIZATIONS
710  DIM B[10,10],S$[1],D$[4],A$[72]
720  D$="NSWE"
730  PRINT
740  PRINT "*** THE GAME OF ZOT ***"
750  PRINT
760  PRINT "DO YOU WANT INSTRUCTIONS ";
770  GOSUB 5000
780  PRINT
790  IF A=0 THEN 1000
800  REM *** INSTRUCTIONS
810  PRINT "IN THE GAME OF ZOT THE TWO (HUMAN) PLAYERS COMPETE TO GET"
820  PRINT "THE LAST BITE OF A RECTANGULAR COOKIE. A MOVE, CALLED A"
830  PRINT "'ZOT', CONSISTS OF TAKING ALL THE STARS IN A LINE BETWEEN"
840  PRINT "A GIVEN SIDE OF THE BOARD AND A PARTICULAR STAR. THE SIDES"
850  PRINT "OF THE BOARD ARE NAMED N,E,S & W (FOR NORTH, EAST ETC.)"
860  PRINT "LIKE ON A MAP. THE POSITION OF THE STAR IS GIVEN BY THE"
870  PRINT "ROW AND COLUMN IT'S IN. FOR EXAMPLE, IF THE FIRST MOVE"
880  PRINT "ON A 4 BY 5 (ROWS BY COLUMNS) BOARD WERE ' E,3,2 ' THE RE-"
890  PRINT "SULTING BOARD WOULD BE :"
900  PRINT "     * * * * *"
910  PRINT "     * * * * *          (ZOTTING FROM THE EAST OVER TO 3,2)"
920  PRINT "     * . . . .          (.'S DENOTE ZOTTED UP STARS)"
925  PRINT "     * * * * *"
930  PRINT "THE STARS IN THE ZOT MUST BE CONSECUTIVE. YOU MAY NOT ZOT"
940  PRINT "EMPTY SPACE UNLESS IT IS BETWEEN THE EDGE OF THE BOARD"
950  PRINT "AND THE BEGINNING OF THE LINE OF STARS YOU ARE ZOTTING"
960  PRINT "(IN THE GAME ABOVE THE SECOND PLAYER MAY NOT ZOT ' S,2,4 '"
970  PRINT "BECAUSE 2,3 IS EMPTY, BUT ' E,3,1 ' IS LEGAL)."
980  PRINT
990  PRINT "HERE WE GO ..."
1000  REM *** START THE GAME
1010  PRINT
1020  PRINT "HOW MANY ROWS AND COLUMNS IN THE COOKIE ";
1030  INPUT R,C
1040  REM *** CHECK IF SIZE IS WITHIN LIMITS
1050  REM *** (NOTE RESEMBLENCE TO LINES 3100 & 4250)
1060  IF R*(11-R) <= 0 THEN 1080
1070  IF C*(11-C)>0 THEN 1100
1080  PRINT "ALLOWED COOKIE DIMENSIONS RANGE FROM 1 TO 10."
1090  GOTO 1000
1100  REM *** CHECK FOR INTEGRAL COOKIE
1110  REM *** (NOTE RESEMBLENCE TO LINE 3160)
1120  IF R-INT(R)+C-INT(C)=0 THEN 1150
1130  PRINT "COOKIE MUST HAVE INTEGER DIMENSIONS."
1140  GOTO 1000
1150  REM *** INITIALIZE BOARD AND AUXILLIARY VARIABLES
1160  FOR I=1 TO R
1170  FOR J=1 TO C
1180  B[I,J]=1
1190  NEXT J
1200  NEXT I
1210  R1=R+1
1220  C1=C+1
1230  N=R*C
1240  P=1
```

```
2000  REM *** MAIN MOVE LOOP
2010  REM *** PRINT THE BOARD
2020  PRINT
2030  FOR I=1 TO R
2040  FOR J=1 TO C
2050  IF B[I,J]=1 THEN 2080
2060  PRINT " .";
2070  GOTO 2090
2080  PRINT " *";
2090  NEXT J
2100  PRINT
2110  NEXT I
2120  PRINT
3000  REM *** FETCH A MOVE
3005  REM *** FETCH BOARD SIDE
3010  PRINT "YOUR MOVE PLAYER ";P;" :"
3020  PRINT "SIDE TO ZOT FROM ";
3025  INPUT S$
3030  REM *** DETERMINE SPECIFIED SIDE OF BOARD
3040  D=1
3050  IF S$[1,1]=D$[D,D] THEN 3092
3060  D=D+1
3070  IF D<5 THEN 3050
3080  PRINT "THE SIDES OF THE BOARD ARE NAMED N,E,S & W."
3090  GOTO 3020
3092  REM *** FETCH CO-ORDINATES
3094  PRINT "ROW, COLUMN TO ZOT TO ";
3096  INPUT I0,J0
3100  REM *** CHECK IF MOVE IS ON BOARD
3110  REM *** (NOTE RESEMBLENCE TO LINES 1040 & 4250)
3120  IF I0*(R1-I0) <= 0 THEN 3140
3130  IF J0*(C1-J0)>0 THEN 3160
3140  PRINT "ROWS ARE NUMBERED 1 THRU ";R;" ; COLUMNS 1 THRU ";C;" ."
3150  GOTO 3092
3160  REM *** CHECK THAT MOVES ARE INTEGERS
3170  REM *** (NOTE RESEMBLENCE TO LINE 1100)
3180  IF I0-INT(I0)+J0-INT(J0)=0 THEN 4000
3190  PRINT "MOVES MUST REFER TO WHOLE NUMBERED ROWS AND COLUMNS."
3200  GOTO 3092
4000  REM *** SCAN ALONG LINE OF ZOT, VERIFY MOVE AND UPDATE BOARD
4010  REM *** SET UP MACHINERY FOR SCANNING
4020  D0=INT((D-1)/2)
4030  S=2*(D-2*D0)-3
4040  I1=S*(1-D0)
4050  J1=S*D0
4060  I=I0
4070  J=J0
4080  REM *** INITIIALIZE SEARCH SWITCH AND STAR COUNTER
4090  S0=1
4100  C0=0
4110  REM *** SCAN OUTWARD FROM MOVE TO EDGE OF BOARD
4120  REM *** TEST FOR STAR
4130  IF B[I,J]=1 THEN 4170
4140  REM *** FOUND A SPACE : SET SWITCH AND CONTINUE SCAN
4150  S0=0
4160  GOTO 4220
4170  REM *** FOUND A STAR
4180  REM *** (IF WE WERE SCANNING SPACE THIS IS AN ILLEGAL ZOT)
4190  IF S0=0 THEN 4300
4200  REM *** OTHERWISE WE INCREASE THE STAR COUNT
4210  C0=C0+1
4220  REM *** ADVANCE SCAN CO-ORDINATES TO NEXT POSITION
4230  I=I+I1
4240  J=J+J1
4250  REM *** CONTINUE SCANNING IF STILL ON BOARD
4260  REM *** (NOTE RESEMBLENCE TO LINES 1040 & 3120)
4270  IF I*(R1-I)*J*(C1-J) <> 0 THEN 4110
4280  REM *** CHECK FOR NULL ZOT
4290  IF C0>0 THEN 4330
4300  REM *** PLAYER HAS ATTEMPTED TO ZOT EMPTY SPACE
4310  PRINT "THE STARS IN YOUR ZOT MUST BE CONSECUTIVE."
4320  GOTO 3000
4330  REM *** MOVE IS LEGAL, UPDATE BOARD ETC.
4340  REM *** (SAME SCANNING TECHNIQUE AS ABOVE AT 4000)
4350  I=I0
4360  J=J0
4370  FOR K=1 TO C0
4380  B[I,J]=0
4390  I=I+I1
4400  J=J+J1
4410  NEXT K
4420  N=N-C0
4430  REM *** CHECK FOR A WIN
4440  IF N=0 THEN 4480
4450  REM *** NO WIN, SETUP FOR OTHER PLAYER AND LOOP
4460  P=3-P
4470  GOTO 2000
4480  REM *** ANNOUNCE THE WIN
4490  PRINT
4500  PRINT "CONGATULATIONS PLAYER ";P;" ON YOUR SUPERB ZOTSMANSHIP !!"
4510  PRINT "(BETTER LUCK NEXT TIME PLAYER ";3-P;" )"
4520  PRINT
4530  REM *** OFFER A REPLAY
4540  PRINT "WOULD YOU LIKE TO PLAY ANOTHER GAME OF ZOT ";
4550  GOSUB 5000
4560  IF A=1 THEN 1000
4570  PRINT "O.K. , SEE YOU FOLKS LATER."
4580  PRINT
4590  STOP
5000  REM *** SUBROUTINE FOR YES/NO RESPONSES
5010  INPUT A$
5020  A=0
5030  IF A$="NO" THEN 5080
5040  A=1
5050  IF A$="YES" THEN 5080
5060  PRINT "PLEASE ANSWER YES OR NO."
5070  GOTO 5000
5080  RETURN
9999  END
```



PCC'S REFERENCE BOOK

Ever been frustrated by the lack of a single source of essential reference information on home computing? Well, we have too, so we finally decided to put it together ourselves, into *PCC's Reference Book of Personal and Home Computing – Spring, 1977.* It will bring together in one place sources for hardware, software, parts and services and for clubs, stores, periodicals, and books. It will also contain many pages of more detailed information on numerous products and services. Useful cross-reference indices help locate the manufacturers of specific products, for example, floppy disk interfaces. Another attraction of *PCC's Reference Book* is the collection of in-depth articles surveying many of the varied aspects of the personal computing field.

Hundreds of sources for products and services, from the tiny one-man shop to the international corporation, are now available for easy reference whenever needed. Micro-computer enthusiasts will be saved time and untold frustration, not to mention the money saved in comparative shopping. And even when the hobbyist is not reading *PCC'S Reference Book* and dreaming about yet another peripheral device for their micro, they will find in its articles a wider perspective and deeper understanding of this new and everchanging field. No one interested in personal and home computing can afford to be without this book of valuable reference information. *PCC's Reference Book of Personal and Home Computing – Spring, 1977* will soon be available for $4.95 from most local computer stores or directly from People's Computer Company.

COMPUTER BOOK BUYERS: LOOK HERE!

The PCC Bookstore Catalog is now going to be a separate publication. It will be sent free to all subscribers of the PCC newspaper. It will appear in your mailbox quarterly, or more often.

The first issue will contain lots of new titles as well as all the old standbys. New books are arriving on the scene faster than we can review them, so your suggestions as to what we should carry are always gratefully accepted. In fact, better yet, turn your suggestion into a review of the book(s) of your choice.

Also, expect two new categories of books:

1. Conundrums and Puzzles to Program – Mathematical and logic puzzles to challenge your program writing skills.

2. Learning – Books to aid in the constant quest for efficiency of learning.

Daniel Rosset
PCC
Box E
Menlo Park, CA 94025

REVIEWS

THE ELECTRONICS PROJECTS NEWSLETTER

REVIEW by Don Inman

Published monthly – September through May
by Robert Delp Electronics
Box 1026
Fremont, CA 94538
$10/year

Though primarily aimed at the educational market, the electronic hobbyist will also be enticed by the low cost construction porjects featured in this newsletter. Each issue contains a featured project with an additional bonus project sometimes included. The publication is approximately 7 pages per issue and includes a parts list and sources for the parts, photographs of finished projects, schematic diagrams, and printed circuit board layouts.

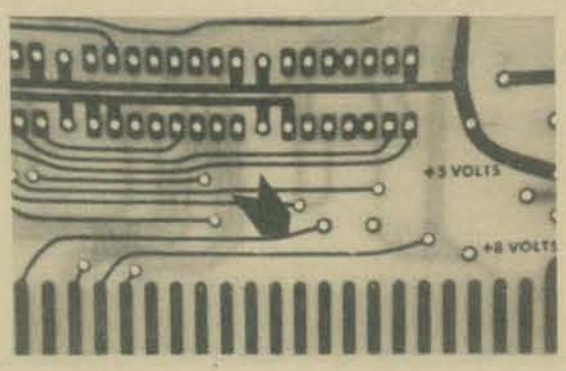
Constructions featured are inexpensive projects with parts typically running from $2 - $2.50. Although the cost of the newsletter ($10/year) at first seems rather high, a school or hobbyist can conveniently build up a library of projects over a period of time. An additional feature which is attractive to teachers is the granting of reproduction rights for classroom use. Also, student activities are suggested for most projects. Time which is usually devoted to scouring electronics magazines for simple projects can be eliminated when the newsletter is utilized. In addition, electronics magazines seem to be devoting their efforts towards more and more complex projects, thus completely ignoring the beginning kit builder.

Typical projects and approximate parts costs featured in past newsletters have been Motor Speed Control - $5, Digital Clock - $15, Talking Light - $6, and Steadyhand Game - $3. Bonus projects have included an economy motor speed control, an alarm circuit for the digital clock, and a radio control for the digital clock.

I would particularly recommend this newsletter to teachers of electronics classes. Several plus features are provided:

- The newsletter is educationally oriented
- Projects are inexpensive (surplus parts can frequently be used)
- Projects are simple, yet useful
- Student activities are suggested
- Plans may be duplicated for classroom use
- Provides a convenient method to build a library of projects

All construction plans are offset printed on 8½" by 11" white paper, one side only. This makes easy reproduction by Xerox, ditto, or other methods.



CALCULATORS/COMPUTERS

by Don Inman

The concept of CALCULATORS/COMPUTERS Magazine was developed to provide practical computing material written for educational purposes. Over the past few years more and more hand held calculators and micro computers have appeared in the classroom as well as in the home. Many schools are exploring their use through experimental programs. It is essential that every student and parent become acquainted with the nature of computers and roles which they play in our society.

Educational computing today lacks proven instructional materials in quantity. Computer software development is expensive. Until the market for such materials becomes sufficiently large, this area will be avoided by the mass market oriented textbook publishers. The development of computer software will be left to the hobbyist and non-profit institutions such as PCC and the schools themselves. Increased computer usage will lead to a corresponding increase in the publication of materials. It is the goal of CALCULATORS/COMPUTERS Magazine to search out material from equipment manufacturers, hobbyists, teachers, parents, students, and other potential sources. We will then edit the material and present it in a form suitable for use in the home or in the classroom. Individualized instruction, with each student working at his own pace and reaching his own level of achievement, has been a goal of education for some time.

We are seeking small modules which are specific, concrete, self-contained, and can be reproduced in sets for direct classroom or home application. Particularly desirable is new, original material which is free and clear of all copyrights. Original material of this nature will be published with the author's by-line and will be paid for at the rate of $10 per laid-out page. The publishers will hold the right to edit such material and present it in the form they consider most practical. Material for use in the classroom should be accompanied by a teacher's commentary or guide. If a particular calculator or computer was used, that equipment should be specifically named and any appropriate model numbers included.

Calculators and computers can support all educational philosophies from conservative to progressive. Basic skills can be reinforced through drill and practice. Special environments and social situations can be created through educational games and simulations. Practical applications of problem-solving skills can be experienced. All of these and more will be explored in CALCULATORS/COMPUTERS Magazine.

As the Editor of CALCULATORS/COMPUTERS Magazine, I believe that computing power exists to be shared, not withheld. Just as a computing facility should provide easy access to equipment, CALCULATORS/COMPUTERS Magazine would hope to provide a vehicle for easy access to and the sharing of software and curricula.

Subscriptions are available from Dymax, Box 310, Menlo Park, CA 94025. Subscription rates:

Vol. I – 3 issues for $4 (May, Oct., Nov., 1977)
Vol. II – 7 issues for $12 (Subscription begins with current issue.)
(Vols. I and II may be ordered together for $14.)

CALCULATORS/COMPUTERS will be published seven times a year: January through May, October and November.

REVIEWS by LeRoy Finkel

INSTANT BASIC

by Jerald R. Brown

For the microcomputer enthusiast or the user of DEC's BASIC PLUS language, there is *finally* a book to teach you BASIC. No longer will you have to struggle through the reference manual provided with your computer. No longer will you have to adapt to a book written for some other dialect of BASIC. *Instant BASIC* was written to teach Altair-style BASIC to beginners using interesting programming ideas and applications that will be easily understood by the home computer programmer. BASIC PLUS users know that the two languages are very similar; thus, this book can be used by them, as well.

The author has a quiet writing style that slowly introduces new ideas in a pleasant non-mathematical context. To offset this style of writing, he has used the zaniest, wildest, graphics available making this book a barrel of fun to read! *Instant BASIC* is an 'active participation' workbook. The author asks you to read it, then do it! The book is designed to be used with your home computer at your side, so you can learn by *doing*.

For those of you who already know some BASIC and now want to learn this dialect, this workbook has handy reference summary boxes that provide a quick-reference summary of most statements (8K Altair BASIC) including differences between Altair BASIC and BASIC PLUS.

You'll like this book. It's designed for beginners, covers most of the language and is fun to use. It's available from the PCC bookstore for $6.00.

YOUR HOME COMPUTER

by James S. White

Your Home Computer could be called a cookbook of home computing since it tells you everything you really wanted to know . . . but were unsure *where* to ask. For those who know little or nothing about computers, it starts at ground zero - *what are computers*, and painlessly introduces the new microcomputing technology and terminology. It tells all about home computer kits, computer stores, and how to use your home computer for fun and profit! It is written in plain language easily understood by the novice. Computers and home appliances are compared to explain how computers work. Loads of pictures help understanding as well.

For those of us that know 'something' about computers, this book brings us up-to-date with micro-technology. I searched for months for a book to serve my non-technical mind, to tell me how microcomputers work and what makes them different from the monsters and minis that I have used for years. I only found books that emphasized microcomputer *programming*. *Your Home Computer* provided the answers I wanted . . . what are micros? How are they different? What's the jargon? What is this about computer stores? Computer kits? TV terminals? What is Altair? IMSAI? Zilog? Intel? What is 8080, 8008, 6800, Z-80? Computer clubs where people meet to talk computers! And give away or trade FREE software! (Heresy.) And most important, 'Hey, what can I do with the thing besides play space war games?'

Well, I got my answers and many more. *And*, the book was fun to read. I'll even suggest to teachers that *Your Home Computer* will serve as an excellent introductory text for students interested in computing (home or otherwise). It's available for $6.00 through the PCC Bookstore.

THE FIRST WEST COAST COMPUTER FAIRE

A Conference & Exposition on Personal & Home Computers

San Francisco Bay Area – Where It All Started – Has Its First Home Computing Convention

7,000 to 10,000 People
100 Conference Sessions
Publication of *Proceedings* Being Planned
200 Commercial & Homebrew Exhibits
Special Interest Social Centers

CO-SPONSORS INCLUDE AMATEUR, PROFESSIONAL, & EDUCATIONAL GROUPS

To Be Held in the San Francisco Civic Auditorium, Northern California's Largest Convention Facility

CONFERENCE SECTIONS ON HOME COMPUTING

Being Planned

- Computer Graphics on Home Computers
- Computer-Driven & Computer-Assisted Music Systems
- Speech Synthesis Using Home Computers
- Computers & Amateur Radio
- Computer Games: Alphanumeric & Graphic
- Personal Computers for the Physically Handicapped
- Computers & Systems for Small Businesses
- Tutorials for Hardware Novices & Software Novices
- Software Design for Personal Computers
- Microprogrammable Microprocessors for Hobbyists
- Optical Scanning for Inexpensive Program & Data Input
- Floppy Disc Systems for Home Computers
- Hardware & Software Standards for Personal Systems
- Seminars for Club Leaders, Editors, Organizers, etc.
- Personal Computers in Education (associated with a University of California short-course)

-- AND MUCH MORE --

PRESENT-WORLD & FUTURE-WORLD BANQUET SPEAKERS

Fascinating Speakers will Discuss the Past, Present, & Future

Banquets to be Held in San Francisco's St. Francis Hotel

Frederik Pohl, Science Fiction Writer, & Lecturer to NASA, NY Academy of Science, etc.
Robots You Can Make for Fun & Profit

Henry Tropp, Smithsonian Institution Researcher in History of Computers, & Mathematician
The 1940's: The FIRST Personal Computing Era

John Whitney, Pioneer Computer Film Maker under grants from Natl Endow. for Arts, Guggenheim, IBM
Digital Pyrotechnics: The Computer in Visual Arts

Ted Nelson, Author, Director of the Xanadu Electronic Literary Network, & Swarthmore College Lecturer
Those Unforgettable Next Two Years

april 15-17, 1977 · san francisco

GET YOUR **FREE** *COPY OF* **SILICON GULCH GAZETTE**

- All the news about the First West Coast Computer Faire
- Plus lots more, just to make it interesting:
  - "Hot news", & raging rumors from "Silicon Valley" (the San Francisco Bay Area)
  - Product announcements, equipment descriptions, hardware & software news and feature articles, etc.
- Details of the *Proceedings* of the Computer Faire
- Just write, & say: "Send me the *Gazette*."

Jim Warren, Faire Chairperson
The Computer Faire
Box 1579, Palo Alto CA 94302
(415) 851-7664

PCC: NEW DIRECTIONS

WHAT READERS WROTE

Whew! I've tabulated the written comments from 250 questionnaires, and find that PCC readers are an incredibly diverse bunch. I've grouped comments into categories to provide some sort of framework for discussion.

Hardware

Readers want information of all sorts on hardware, especially low cost stuff. Practical and detailed information is wanted in a variety of areas.

- Equipment reviews that cover the types of systems available, pros and cons of each, including kit assembly time, frequency of hardware problems, etc.
- Guidelines on how to decide when you can't handle some hardware problem – and what to do then
- Articles to remove some of the mysteries of hardware purchase/construction/debugging/maintenance for people who are software oriented
- Info on hard copy CRT terminals that are cheap or low cost conversion
- Schematics, circuit analysis, logic analysis

And while it was a fine idea, I regret to have to announce that PCC will *not* be providing free hardware as more than one reader suggested.

Interfaces

Several readers made requests for specific interfacing information. I'm sure the list could be added to by almost every one of you. So far requests include

- adapting an IBM selectric to serve as I/O for a microcomputer
- how to interface an Altair to a teletype
- how to adapt Altair BASIC for Polymorphic's or Processor Tech's video interface
- how to add an ASCII keyboard to an Altair parallel input board
- how to put Altair (or other) BASIC on PROM boards
- OCR home key terminal to tape

In addition, several readers expressed interest in learning more about techniques for networking several computers.

Software

Readers want more sophisticated articles on software, including info on design, implementation, troubleshooting, maintenance, and what's on the drawing boards. A number of readers want to learn more about operating systems, compilers, machine assemblers, cross assemblers. System simulation, multi-user systems, parallel processing, and human interface problems were also mentioned.

High-Level Languages

Many readers want to learn about new or unfamiliar programming languages. Some of the languages specifically mentioned include FORTRAN, APL, PASCAL, TRAC^{R}, C, PL/1, LISP, LOGLAN, and SNOBOL. General information and comparative descriptions are desired as well as code or inexpensive sources for compilers/interpreters. Implementation on micros was of special interest.

Applications

Lots of PCC readers have their own micros, and great hopes about things to do with them if we look at the desired applications. A partial list includes:

- Graphics, a high priority item for many; computer art was also mentioned
- Text processing, another high priority
- Artificial intelligence game/simulation programs were also requested by quite a number of readers
- Plotting info
- Software engineering techniques
- Mini/micro software exchange
- Computer-aided design
- Mailing label systems
- Inverted index systems
- Programs to help automate home bookkeeping activities
- Programs for data storage and retrieval filing systems
- Very small business software (accounts, inventories, lists)
- Software for reading printed material via TV camera
- An emulator for the HP-35 or SR-52
- Robotics
- Exotic applications using opto-isolators, A/D and D/A convertors
- Sampling of experimental data
- Votrax voice generators
- Control of mechanical devices
- Mail order computer keypunch services and computer reports by mail

Education

The issue of education was pretty much covered in detail on the questionnaire, but there were some write in comments anyhow. There were several requests to look at various ways to use large systems for educational purposes. PLATO IV was mentioned, as was instructional management (assessment, diagnosis, prescription et. al. for teachers' use, for counselors' use, and for self-analysis by students.) The ever-growing use of micros in schools is something a number of readers wish to know more about; one person requested information on microcomputer products of interest to people teaching computer education courses. Finally, PCCers want more tips on how to turn on kids and others to educational uses of computers.

Games and/or Listings

While a few readers wanted fewer listings (suggesting that they could be ordered if desired) most readers wanted more listings, and more readable listings. While interest was expressed in games, there were requests for other types of games than those usually seen in PCC: more games of the 'life' type by Conway, fewer grid-oriented games, more space games, and more games of interest to adults. Listings of educational non-game programs were also requested.

Format/Structure

A couple of people requested better organized layout, and asked that we consider going to a magazine format. Better quality printing was also mentioned, as were more issues per year.

Other

Well, there were several requests for centerfolds (watch it, the editor is a woman – wanna send me *your* photo?) and one irrational request from a member of the PCC Board of Directors for financial stability, but in general, 'other' consists of undiluted praise ('I like you the way you are', 'Just more of the same, please', etc., etc.) or individual comments that didn't fit nicely into any other (oops) category. For example, there was a request for a list of names and addresses of people willing to help get a system up, another for more emphasis on programmable calculators, another for information on human engineering, with an emphasis on ease of use in man/machine interfaces.

This is also the category where opposites often met: some readers like fantasizing the future and would like to see more of this, perhaps as science fiction stories; others emphasize the need for fewer flights of fancy and more down-to-earth practical applications. Oh well, we can but try to please all the people all the time . . . .

EDITOR'S REPLY

The readership survey has helped clarify just what readers want from PCC: we hear you. Articles will be presented from the viewpoint of those who know relatively little about the subject area. In some areas, a series of articles may be needed to educate ourselves to a reasonable point – e.g., solutions to most interfacing problems require at least a rudimentary familiarization with the systems you're trying to put together.

We are in the process of developing mechanisms to solicit the type of articles you want, and you our readers may well be able to help in this area. Take a look at the suggested list in the next section of this article. It is by no means exhaustive, and should be used mainly to spark ideas.

Recently we've had less to offer than many of you would like on the subjects of hardware and interfacing, but we hope to add more such articles gradually. Meanwhile, we'll continue to try to include new product announcements that relate to your requests (e.g., see how to get more info on interfacing a Selectric typewriter to a microcomputer under Announcements on Page 47) and relevant reviews (see Don Inman's review of the Electronics Projects Newsletter, Page 40).

Software and high-level languages have been and will be approached from a number of different angles. The most recent Tiny BASIC and PILOT articles can serve either as introductory material for beginners, or as tools for those who wish to teach beginners. At the same time we hope to to stimulate interest in helping us define standards for easy-to-learn and easy-to-use high level languages which

may be implemented on micros. We've begun to line up potential authors to provide overviews-for-beginners on a number of different languages and hope to soon be publishing the results of these efforts. And for those of you interested in learning about assembly programming, don't miss Don Inman's series on the Data Handler; Part 2 begins on Page 16 of this issue.

How did you artificial intelligence enthusiasts and/or robotics fans like the last issue's article on pet robots? We expect to be publishing more info from the United States Robotics Society and we're also trying to line up some more nuts and bolts applications articles and, most especially, programs.

We'd like to make it possible for anyone with access to a computer to create her/his own educational materials. We'll give you all the help we can in implementing high-level languages suitable for constructing such materials. For starters, the Z-80 assembly code for the language PILOT appears in this issue. We will also publish chunks of code that can serve as templates for writing programs suitable for varying educational objectives. For example, we will publish examples of how to write the following in PILOT:

- Free-flowing conversational programs, which can be used to entertain, or to discuss whatever you like. Just what the user inputs is often used to control the topic under discussion. Such programs are not aimed at getting 'right answers' but at stimulating students, rather as in a Socratic dialogue. Variations on this type of program could be written to request information for your taxes, etc.

- Guessing game programs have right answers. They may also be set up to allow a certain number of attempts at getting a correct answer, to score the student, etc. Such a structure may be used to encourage children's reading skills in a game-like setting, or, at the other end, perhaps, in a drill and practice fashion. Again, the content will vary at the whim of the author. Teach math, chemistry formulas, nutrition, or Swahili nouns. Beware tho, once you get into building these materials, you'll want to have access to lots of storage.

- Interactive-story programs are really just a more interesting way of looking at a form-filling program. Kids and adults alike get a surprising charge out of generating stories. See, for example, the Goldilocks program with the PILOT article. This approach can be used in many form-filling applications as well.

Finally, we'll give tips on how you can teach others how to write such programs. Some such programs can be built even by 5 and 6 year olds.

We'll be paying closer attention to all listings that we publish. In general we want to improve documentation. We will try to make sure that programs we publish are both well proven fun and thoroughly documented and debugged, but we'll need your help to accomplish these goals. As time and money allow we also hope to annotate various listings that we publish, to provide readers with comments and perhaps criticism of the programming style (or lack of it?) illustrated by various listings. Since so many readers are using listings to pick up pointers on programming, we decided this might be a useful way to enhance the value of the listings. Of course this means someone at this end must be able to read and understand your code – just having evidence (say from several runs) that the code works won't be adequate. We offer the PILOT and ZOT articles in this issue as illustrations of the directions we plan to pursue.

You may have noticed by now that our layout style is changing. We'll be making changes, some gradually, some not, over the next several issues. We're also investigating changing paper and/or going to a magazine format. And would you believe a new name? We're considering 'Peoples' Computers' . . . . . watch for advance notice of our new form at the April 15-17 Computer Faire in San Francisco.

ARTICLES WE'D LIKE TO SEE

This far-from-complete list is provided as suggestions only. The order in which items are listed is of no significance. Some topics as specified are worthy only of a brief discussion; other could serve as a basis for a series of articles. In all cases we wish to present material suitable for those who know little or nothing about the area under discussion.

In addition to articles, we hope to publish reviews of books relevant to our readers' interests. We also cheerfully mention articles of interest found in other publications.

Unfortunately, we can offer no monetary incentives to would-be authors. (To say we're on a shoe-string budget is an overstatement.) However, if we publish an article of substantial length, we will thank you with a complimentary one-year subscription if you so desire – or extend your current subscription.

Hardware

- How known system problems can be corrected (e.g. 'How I built a $3 widget to correct the flakey whatsit in Zitel's 8080-based system.')
- Maintenance tips: specific, general, what to do when all else fails
- Details on individual systems running now and in the works
- Overview of computer kits: assembly time, frequency of hardware problems, amount of memory required for various tasks, etc.
- Information on a cheap hard copy CRT terminal
- Information on parallel processing and pipeline processing (useful for image processing, music, and other applications where speed is needed.)

Interfacing

- How to adapt an IBM Selectric typewriter to serve as I/O for a microcomputer
- How to interface a teletype to various micros
- How to adapt Altair BASIC for Polymorphic's or Processor Tech's video interface
- How to add an ASCII keyboard to an Altair parallel input board
- Networking

Software & Languages

- Comparing algorithms to do specific tasks (e.g. sorting)
- Overviews of various langauges, including strengths, weaknesses, where to get more information, characteristics of systems on which they're usually implemented; languages mentioned by readers include APL, PL/1, PASCAL, TRAC ®, FORTRAN, LISP, SNOBAL, BASIC, and SMALLTALK.
- Articles on assembly language programming
- Meaty articles on software design, implementation, debugging, documenting
- Articles on the ins-and-outs of compilers
- Information on operating systems
- Comparison of existing and proposed computer languages

Applications

Both in-depth background articles and well documented debugged programs are of special interest in this category.

- Graphics, computer art
- Text processing
- Plotting
- Small business software
- Mailing label system
- Personal and small business tax systems

Education

- Source code for easy-to-learn and easy-to-use high level languages
- Well documented, debugged instructional materials on all sorts of topics and in all sorts of formats (drill and practice, tutorial, etc.)
- Information on what's available in the computer education arena, including home educators, schools (including elementary to post graduate and vocational training), research, industrial and military training, community computer centers and information systems and companies that sell computer education materials.

Games

- Well proven, fun, documented, debugged games
- Games of interest to adults such as those suggested in last issue's Don Quixote Starship

Other

- Discussion of man/machine interface problems
- Articles on programmable calculators
- Articles on artificial intelligence
- Information on robotics

SUBMITTING ITEMS FOR PUBLICATION

LABEL everything please, your name, address and the *date*; tapes should also include the program name, language and system.

TYPE text if at all possible, double-spaced, on 8½ x 11 inch white paper.

DRAWINGS should be as clear and neat as possible in black ink on white paper.

LISTINGS are hard to reproduce clearly, so please note:

- Use a new ribbon on plain white paper when making a listing; we prefer roll paper or fan-fold paper
- Send copies of one or more RUNS of your program, to verify that it runs and to provide a sense of how things work – and to motivate more of us to read the code. RUNS should illustrate the main purpose & and operation of your program as clearly as possible. Bells, whistles and special features should just be described in the documentation unless they're particularly relevant. See the example using ZOT in this issue.
- Paper tapes of both the program and runs can provide us with a way to make our own listing if we need to. Then, if you give us permission, we can let CCC (Community Computer Center) sell your program cheaply via paper tape, to further the spread of inexpensive software. Finally, if we are so lucky as to have access to a system on which your program runs, we can try it out ourselves.
- Make sure your code is well documented – use a separate sheet of paper. Refer to portions of code by line number or label or address please, not by page number. When writing documentation, keep in mind that readers will include beginners and people who may be relatively inexperienced with the language you're using. Helpful documentation/annotation can make your code useful to more people. Documentation should discuss just which cases are covered and which aren't.
- If you send us a program to publish, we reserve the right to annotate it (don't worry, we won't publish it if we don't like it). See the PILOT article as an example.

LETTERS

I read in your Jan/Feb issue of 77 about the Don Quixote Starship project. I am interested in space games, especially in a non-competitive mode. It seems to me that the game may be made interesting by having a variety of goals, distinct for each player. The game is won when everyone has attained their goal. Limited resources make the attainment of any goal impossible without cooperation. Perhaps a universe exploration could contain as goals and players a mapmaker, a technician whose primary task is to improve the poor communication net between players, a trader-merchant whose goal is to transport raw materials and produced goods to where they are needed, perhaps a doctor and a zookeeper, a farmer, and an industrialist.

What is the Plato system? How can I get in contact with it/them? The blurb on their rumored everlasting 100 light year cubical universe fascinated me but I don't know how to follow up on it.

Stuart Smith
Computersmiths
Box 755
Meadow Vista, CA 95722

We'll try to publish an article on the PLATO system in the near future, and hopefully by then we'll have tracked down the rumor about the simulated universe game.

I would love to see PILOT or tiny PILOT in a form which requires no software knowledge (beyond using tiny PILOT itself) and only enough hardware knowledge to get it off the type and into memory.

Also – there may be a few others like me, who would like a word-oriented language, but are not educational-use oriented. Those "cheap" computers (cheap to a school, but not to me) are available because of the hundreds (?) of hobbyist users out there. Get them interested, and the demand created will see versions of PILOT and tiny PILOT for all the "cheap" machines.

Lastly – your non-exhaustive list of cheap computers in the tiny PILOT article in vol. 5, no. 3 PCC did not mention two manufacturers of interest to soft-ware oriented (as opposed to hard-ware oriented technician/tinker types) persons, i.e. the APPLE computer and the Processor Technology SOL. Both are "all on one board" types, including interface to the outside world, and allow you to get a system up with minimum fuss.

Michael Morris
4927 Asteria Street
Torrance, CA 90503

Here is an opportunity for some *PCC* or *DDJ* readers who have software questions. My background is strong in software design (3 years of programming including system programming on small computers; an MS in Computer Science) but I'm a helpless neophyte when it comes to electrical design. Although I know Ohm's Law and the difference between a resistor and a capacitor, I would be hopelessly lost trying to read or understand the schematics of an amplifier. The only testing (of electronic devices) that I've done has been under the "plug and smoke" method. (I can read logic diagrams–those limited to flip-flop, gates, and invertors.) I would very much like to begin work on a homebrew system but I have a bad case of cold feet.

I'm especially interested in computer controlled music and interactive text editing.

What I would like to do is "trade" my software understanding for some answers to hardware questions. Starting with the general ones below and progressing to more specific ones later. In exchange, I am willing to attempt to answer any readers' questions about software design, including a general willingness to write (untested) programs.

As for me: Where should I start? Know any good "How To" books for a bungling pre-amateur? What test equipment is necessary? (Limited budget!) Can I save money on test equipment? (Kits, rentals–where?) Are there some sub-projects on which I could "teethe" that could easily become an essential part of a later system?

As Bob Dylan said: "I'll let you be in my dream, if I can be in yours." Let's exchange questions, answers, and ideas. My address is:

Terry E. Weymouth
4702 Beau Bien Lane East
Lisle, Illinois 60532

(Please...a price tag of $20 gives me a headache and I get the hardcore blues around $100.)

Oh, how I love *PCC* and *DDJ*! Keep it up.

I have read letters from other people in your newspaper about starting HP-65 clubs. Although the HP-65 is an awfully good calculator, it is also awfully expensive. I have recently purchased a SR-52 and I feel it is comparable to the HP-65, and I would like to get a SR-52 club started. I think maybe correspondence between the HP-65ers and the SR-52ers would also be appropriate since programs from one calculator can be converted to the other without too much trouble. Anyone interested?

Corey Ashford
2604 N.E. 61st
Portland, Ore. 97213

Wanted: Games! I want to trade game programs in BASIC with anyone in the country. Especially Star Trek games.

I have heard a rumor that a PCC "station" will be setting up in Cincinnati. Is this true? If so, where and when?

John Baylis
4850 Drake Rd.
Cincinnati, Ohio 45243

A PCC station? Interesting notion, but nothing we've heard of. Keep us posted!

Is it possible to obtain a complete set of the *Fortran Man* series through Volume 4? I would be willing to pay for back issues containing the series, and for reprints or xerox copies of installments that were in issues that are no longer available.

Thanks.
Brad Baldwin
Box 2907
Garland, TX 75041

P.S. – I receive BYTE magazine and several more trade publications, and I think yours is the most creative of all. Keep it up. By the way, who does the drawings? They're really fine.

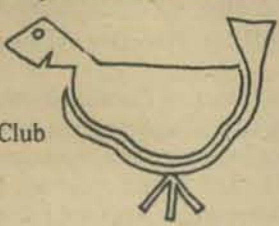
Ann Miya, a freelance artist, does the illustrations. We're exploring the possibilities of putting out an F-Man comic book.

With regard to Lee Felsenstein's idea for 10-watt FM broadcast stations as one-way data buses: I don't think this will work in this form. A 10-watt station has a reliable range of about 2½ miles. The number of channels available *in each area* is limited by the FCC. This is for non-commercial as well as commercial. One could get into a situation where the only available channel is in the commercial part of the band, and engineering and program requirements for that frequency are much stricter.

I would give a serious look at two-way data communications in the 1215-mHz amateur band, or in the business communication segments of the 470-890 mHz TV band, or one of the educational closed-circuit TV channels. There's an outfit in the bay area that makes reasonably priced ham gear for this band. A good article appeared in the December 1975 issue of QST. Similar equipment can be developed as add-on gear to 144 or 432 mHz ham gear or public-service surplus or even police0band scanner receivers. Anyone who can build a Data Handler and have it work with minimum de-bugging can get a Technician Class ham license with very little trouble. There is a nationwide push on not only to get more hams licensed but to expand usage of the amateur bands between 1215 and 10,500 mHz. Since the engineering restrictions are minimal the service area problem is also minimal. You could pilot and debug the system on the ham bands and then (having found appropriate technical standards, move on to a co-op or non-profit installation on a commercial channel. One reason for staying above 1215 mHz is there's more space to play with, and less chance of interfering with someone else on an adjacent channel with high-baud-rate transmissions.

Mike Fern
1046 So. Westlake No. 1
Los Angeles, CA 90006

I am secretary of the Lowell High School Computer Club, and ever since school started we have been trying to get a terminal to use. There is a fine computer at Wilson High School which we could hook into through a phone connection, but we have no terminal; this makes things difficult. Half of our original members have already quit, and the rest are threatening to walk out en masse if we don't do something soon. We have tried to raise money, but at the rate we're going we should get our terminal sometime late 1988. We would greatly appreciate it if you could help us, because we're getting kind of desperate. Can you help us? And if so, on what terms? I'm sure we'd be glad to order a few paper tapes to help defray the costs; some of those programs look absolutely fascinating. I hope to hear from you soon, and I wish you all a good day.



Sincerely,
Chris Wesling
Secretary, Lowell Computer Club
449 Ninth Avenue
San Francisco, CA 94118

PCC has no $ available for such efforts; any angels out there in readership land? Also, it's CCC, not PCC, that sells paper tapes.

I am willing to supply descriptions or listings to anyone as long as they supply an envelope and sufficient postage. The description is 10 pages long and the program listing and program description is 30 pages long. I am also willing to exchange game listings for programs written in BASIC, FORTRAN, APL or almost anything else.

This is the first version of the game, and I have no doubt that numerous changes will be made. I am also thinking about rewritting it in ALGOL or PL/1 in the near future.

Mail should go to either my school address (Sept. to May): Dave Warker
Box 5739
River Campus Station
Rochester, N.Y. 14627
or my home address:
715 North St.
Egg Harbor, N.J. 08215

There is a star trek game, written in BASIC, called TREK 73. After playing this pseudo real-time game, I decided to try to write a game of my own. The result was approximately 1400 lines of FORTRAN code, in its present form. The basic game setup is as follows:

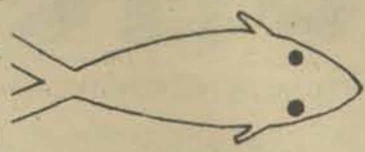
The player controls a ship with phasers, photon torpedos, warp engines, etc. He plays against from 1 to 5 Klingon ships which are equipped similar to his, and are controlled by the computer. They presently have no strategy but that will be added soon. The game is pseudo real-time, ships and torpedos have a velocity and take time to get to wherever they're going. There is a rudimentary rating system and various other nicities built in. The game ends when either the Klingons or the player lose all their shields, or if the player terminates the game with a special command.

The program is written in "barebones" FORTRAN, so the program should compile on any computer that supports fortran with only a few changes.

I am intrigued by this game of Dungeons and Dragons. How can I go about joining this organization?????? Is there any group in my area that you know of? (or your area, or ANY area?) Whether man, myth, or Dragon, it seems to be the thing to do.

How does one go about joining your organization of Pen Dragons? Is this open to anyone or do I have to have fiery breath? (only after a hot pizza if you must know). I'm sweet, cuddly and don't make messes in the house. How about it?

After rambling on, I have but one thing to say. Keep on Dragon (pun). Seriously thou, I like your magazine (*PCC*, I haven't seen Dr. Dobb's), so keep up the good job. You aren't alone.



Bradley J. Q. Johnson
The Wizard of Q
714 N. 6th St.
Lake City, MN 55041

Publications involved with Dungeons and Dragons games are listed on page 42, PCC Volume 5, Number 3. And see this issue regarding guidelines for submitting stuff for publication.

---

1) I like the IL (intermediate language) approach. It is very similar to the method that has been evolving for some software I'm working on (a monitor and assembler for the 6502), and now I begin to wonder if I cannot make the two compatible, i.e.: use the same IL interpreter for Tiny BASIC (TB) and the monitor/assembler. There are some different machine language (ML) calls, but there is plenty of room to define more in TB's IL (and almost unlimited room in Whipple & Arnolds version). Will let you know how this works out. It seems to me that a well developed master IL (MIL) might be possible which would allow development of editors, executives, assemblers, languages, maybe loaders, etc. Is this blue-sky?

2) How much slower should the two level approach be as compared to a one level approach? (Factor of 1.5,3,10, 40?) Any rough ideas?

3) Calculator chip extended function units are too slow!

4) I agree: 'nuff BASIC, SMALL PASCAL would be of much interest. CASUAL and TINY HI are intriguing.

5) More references to trade & professional sources (literature). (speaking of which, where does one find the planet Pern?)

6) Re using an acoustic coupler for cassette tapes: this may not work very well. See the analysis in *The Computer Hobbiest*, No. 5 (Box 295, Cary NC 27511). Briefly, the "scuzziness" of a cassette recorder and of a phone line are of different natures. Also, who needs another 300 bps cassette standard? One major problem with the KC standard is that it's too slow, so everyone wants to use something faster for themselves (and two cassette interfaces is awkward to ask of folk). And for higher bit rates, you can be pretty sure that high speed modems (*IF* they work on a cassette at all) will be much more expensive than other (self-clocking) techniques, e.g.: Digital Group & Tarbell. And a final note for those who go ahead anyway (which I encourage as long as nobody forks out too much dough *until* this is tested): standard acoustic couplers transmit on one frequency pair and receive on another pair, i.e.: they won't read what they write. On *some* modems, the "self-test" mode will allow transmitting and receiving on the same pair. I believe the Motorola MC14412 universal modem chip will allow this, also. (Likewise the MC6860 of the 6800 family has self-test.) Good luck! It should work (at 300 bps on a decent cassette unit). Exchange may be pretty hard tho.

7) Good articles on double density for floppies in Sept., Oct., and Dec. issues of *Computer Design*. Re HLL, see *A General-Purpose Macro Processor as a Poor Man's Compiler-Compiler* by Andrew S. Tanenbaum in IEEE trans. on software eng., Vol. SE-2, No. 2, P. 121 (June 1976). Of special interest is his description of SAL, a PASCAL-like systems programming language. I think it could be Tinied cleanly.

Zhahai Stewart
P.O. Box 1637
Boulder, CO 80306

Desperately looking for a BASIC Interpreter to run on my KIM-1 System. Will gladly pay! At your mercy!

Edward L. Pavia
127 Sugar Maple Drive
Rochester, N.Y. 14615

I've got a four-voice, polyphonic tone generator subroutine for the 8080 which I'm trying to sell. Actually its the great-great-great grandson of the first one I wrote, and I think this one's finally good enough. In fact, I was calling myself all kinds of genius the night I wrote it! Here's the spec's:

Range: Four octaves of chromatic notes centered around middle-C. (With a 2 MHz CPU clock)

External parts required: One latched four bit parallel TTL output port, four fixed uncritical resistors, one electrolytic cap (uncritical) and an amp.

Adjustments to hardware: Set volume and tone on your amp to your liking. Maybe (but I doubt it) change some parts if your amp has too low gain or you were way off on the original values.

Frequency accuracy: +/– almost ½ a chromatic interval on the tippy-top notes; error decreases in proportion to the frequency, so the bass is very reassuring even if the top half-octave is "rinky-tinky".

Tuneability to other instruments: Maybe, by putting in NOP's or something.

Framing error (pulse width modulation): No.

Intermodulation distortion or other interaction between notes: None, unless your amp is really bad and you have to reduce the values of the resistors to almost nothing to get the volume you want. Each voice comes out on its own bit of the four-bit port (you could have stereo output – even quad!)

External timing circuitry required? Nope. It plays from zero to about four seconds of sound with 65K possible durations in-between.

Waveform control: None.

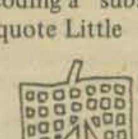
Envelope control: On-off on each note. The four left-over bits on the output port could trigger some simple clipping type envelope shapers...

Memory requirement: 58 (decimal) bytes of fast, static RAM. (1000 nanosec static rams would decrease speed and frequency by about 30%). Four one-byte constants must be "stuffed" into the program by the calling routine to change notes. (Five if you want trigger outputs.) Also, the table to convert chromatic note-numbers into timing constants is 52 (decimal) bytes.

Thumping: Slight thump when a note goes "on" or "off". Otherwise, if voices are sustained, there is a slight "wivvle" caused by the pause as the driving routine looks up the next note of music.

Selling arrangements: $20, and you can do anything with it (except copyright it!); $2 if you promise not to distribute copies of it. I won't accept any payment which doesn't turn into cash quickly – cash is fine if you want to risk it! These prices are contingent on demand. You get source listings of the tone generator routine and a "dumb driver" program, object listings, table of timing constants, list of a song for the "dumb driver" (those last three in octal, hex or decimal – please specify), circuit for getting the music out, and some comments. I assume you know how to relocate and modify object programs.

Demonstration tapes: If you send a cassette in a prepaid mailer with 50¢, I'll record on it whatever music I've whipped up by then, and answer any questions you record on the cassette. The same goes for reel-to-reel tapes, but my recorder is stereo. (Also indicate speed.) Right now I'm coding a "subset" of Bach's G-minor fugue. ("the quote Little unquote") That's all, folks.



Steve Witham
168 Painter Road
Mecia, PA 19063

ANNOUNCEMENTS

VT-4800 VIDEO TERMINAL

The standard features of the VT-4800 include:

- Its own 4K bytes of RAM
  Expandable to 16K bytes
- Complete cursor control
- Direct Cursor addressing
  Allows keyboard or software to position the cursor anywhere on the screen
- Character read at cursor position
- Scroll up and Scroll down
  Allows up to 16K of RAM to be scrolled through before any data is lost
- Five clearing controls
- Page increment and decrement
- Character enchance
  Displayed character can be white on black or black on white
- All 32 control functions decoded and strappable
- Standard RS-232, TTL serial I/O, and TTL parallel I/O
- Selectable BAUD rates
  110, 300, 600, 1200, 2400, 4800, 9600
- Interfaces to any ASCII keyboard
- Composite and separate video outputs available

The VT-4800 is available primarily in kit form in any configuration from single boards to 100% complete kits. Assembled and tested models can be purchased for a standard assembly fee. Please consult current price list for detailed descriptions of options available.

Video Terminal Technology
P.O. Box 60485
Sunnyvale, CA 94088
(408) 255-3001

NEW ANALOG SWITCH SERIES

A new series of analog switches developed by National Semiconductor Corp. combines bipolar and JFET technology, producing the Industry's first single-chip quadruple JFET switches. Combined with this new monolithic process is a unique circuit technique that allows the new analog switches to maintain a constant resistance over the wide analog voltage range of +10 volts.

Designated the LF11331, LF11332, LF11333, LF11201, and LF11202, the new analog switches are designed to operate from minimum TTL input levels and feature a break-before-make switching action. The LF11331 contains four normally-open switches with a common disable pin that opens all of the switches in the package. The LF11332 contains four normally-closed switches with a common disable. The LF11333 contains two normally-closed switches and two normally-open switches with a common disable. The LF11202 has four normally-open switches.

All of the new analog switches feature constant 'ON' resistance for signals up to +10 volts and 100 kHz, high open-switch isolation at 1 mHz of 50 db, off-state leakage of less than 1 nanoamp, and can handle small level analog signals up to 50 mHz. They are intended for use in applications where a dc-to-medium-frequency analog signal is to be controlled by a TTL, DTL, or RTL logic signal such as in computer controlled audio switching systems and in digitally controlled process control systems. The devices operate from +15 volt supplies and switch a +10 volt signal.

Three operating temperature ranges are offered for all of the analog switches. The LF11331, 2 and 3 is intended for operation over the -55 to +125°C military range and is available in a 16-pin ceramic DIP. The LF11201 and 2 series is intended for operation over the -25 to +85° C range and is available in either a 16-pin ceramic DIP or a 16-pin Epoxy B DIP. The LF13331, 2 and 3 series is a commercial grade version which operate over the 0 to +70° C range and is availalbe in Epoxy B DIP. In quantities of 100, the LF13331 sells for $2.85 each. Delivery is from stock.

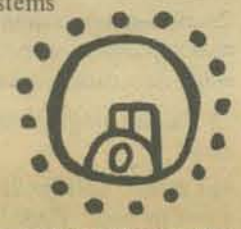
COMPUTER ART COURSE!!!

The computer as a tool for the artist will be taught at De Anza College in Cupertino, California this Spring Quarter, April–June, 1977. (Art 22, 3 units, Wednesdays 2:30-4:30 p.m.) Gregory Yob, known to some of you for WUMPUS and MANDALA will be teaching. There are no prerequisites and both non-programmers and home brewers are welcome.

CONVENTION

15th Annual Convention
Association for Educational Data Systems
April 25-29, 1977
Green Oaks Inn, Fort Worth, Texas

National Headquarters:
1201 Sixteenth Street, NW
Washington, D.C. 20036
(202) 833-4100



CONFERENCE

1977 COMPUTER USERS CONFERENCE March 25, East Texas State University, Commerce, Texas. Session topics are "Large Systems" and "Mini and Micro Systems."

In addition to keynote speeches given by C.A. Conover and Harvey Cragon, panel discussions will be held by industrial and educational representatives concerning needs and trends in the respective computer usage areas.

Fees: $20 ($10 for students). For further information contact Donna Hutcheson, Computer Users Conference Coordinator, East Texas State University, Department of Computer Science, Commerce, Texas 75428.

Telephone: (214) 468-2954

2ND TRENTON COMPUTER FESTIVAL – 77

WHAT: $4.00 General Registration; $2.00 Student
Huge Hardware & Software Flea Market
Amateur Computer Club Convention
Special IEEE Conference on Consumer & Hobby Application of Microcomputers
Plus exhibits, displays, technical talks, contests, home computing, prizes, etc., etc., etc!

WHEN: April 30/May 1, 1977 beginning at 10:00 a.m.

WHERE: Trenton State College, Route 31, Trenton NJ

FOR FURTHER INFORMATION: Call Jaci DiPaolo (609) 771-2487 or write Trenton Computer Festival, Trenton State College, Trenton, NJ 08625.

WORLD GAME '77

The 8th Annual World Game Workshop will be held this summer in Philadelphia at the University of Pennsylvania in conjunction with the University City Science Center, the Franklin Institute, the International House of Philadelphia, and Buckminster Fuller. The program is produced by Earth Metabolic Design, Inc. and consists of two stages:

STAGE I: Planetary Planning Symposium (June 19-25)

The Planetary Planning Symposium will be a week-long schedule of morning and evening lectures with alternative afternoon seminars on various topics related to global planning. The lecture series will feature distinguished scientists and humanists who will present their viewpoints and theories concerning critical world-wide problems and their possible solutions. In previous years, these speakers have included: Buckminster Fuller, Ian McHarg, Russell Ackoff, Howard Odum, John Platt, Ervin Laszlo, Hazel Henderson, Stuart Brand, Erich Jantsh and many others. This symposium will be held before a large multidisciplinary audience during the week of June 19th to June 25th.

STAGE II: Design Science Laboratory

The Design Science Laboratory is designed to provide a more in-depth working experience in comprehensive planning for human needs. It will consist of several long-range planning projects facilitated by a variety of experts at their place of work. The orientation program for this process will be held on June 26th and 27th in Philadelphia and the projects themselves may last as long as six to eight weeks in several locations besides Philadelphia.

The purpose of the entire workshop is to explore and design alternatives for better meeting the life-support needs of all humanity. It is centered around Buckminster Fuller's metaphor of "World Game", which describes a process of exploring alternative planning and design strategies for providing progressively higher standards of living for all humanity.

In all of the past World Game Workshops, participants have been organized into planning teams to develop strategies for better meeting human needs in an environmentally sound manner. The participants in World Game '77 will explore the potentials of comprehensive planning as a means of solving critical world-wide problems. This requires a multiplicity of policies, strategies and designs, tested and developed by as many people as possible. In this context, World Game '77 will be an educational experience in which people can learn to become participants rather than spectators, acquiring the confidence and ability to initiate positive actions upon specific problems in the world today.

There are a limited number of scholarships available for the workshop. It is possible to attend the week-long planetary planning symposium alone for a tuition of $200, or the entire workshop for a tuition of $350.

For further information and application write:

WORLD GAME ' 77
University City Science Center
3500 Market Street
Philadelphia, PA 19104

MORE THAN YOUR MIND CAN HEAR

"Unplayed by Human Hands – A Computer-performed Organ Recital" is the first record of its kind. It is the five-

year product of what began as a computer hobby, some time before microprocessors became available. For that reason, the system is implemented on one of DEC's original refrigerator-size PDP-8 systems. In the future there's no reason the system cannot be duplicated on a microprocessor.

It's $8.00 for one, $12.00 for two, and less in larger quantities.

Dr. Prentiss Knowlton
Computer Humanities
2310 El Moreno Street
La Crescenta, CA 91214

□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□

THE COMPUTER PROGRAM

This is a proposal for a radio program which will be of interest to the home/amateur/personal computing hobbyist. "The Computer Program" would be a radio program of interest to this constantly growing common interest group.

If all went as scheduled, the first broadcast of "The Computer Program" was on WBUR, 90.9 FM, Boston, January 22, 1977, 11:00 a.m. to 12:30 p.m., live, with calls from listeners.

The planned format is:

1) A guest will speak on a topic of his choice.
2) The moderator/host will ask questions related to the topic.
3) Listeners may call with questions for the guest.
4) Announcement of a bibliography/resource list related to the guest's topic, including books for beginners, films, publications, events related to the topic, experts, manufacturers and distributors of related equipment or materials.
5) Request for listener response to the program in general, and the guest's topic in particular.
6) Computer Reports – like Consumer Reports, you will learn which computers users think are best (tis better to be learned than burned!)
7) General news items of interest to the home computer hobbyist.
8) Raw Random Rumors
9) Musical open and close? Any suggestions, besides DAISY as a close?
10) Announcement of the next guest speaker.
11) Credits

The program can vary in length from 1/2 – 2 hours, with space for advertising if broadcast on commercial stations. Parts of each program, such as the guest speaker, or general news, could be broadcast separately. COMPUTER STORES should note that this may be a good way to tell the world about your products and services! Write or call for information about how to have this program braodcast in your area.

Richard Gardner
The Computer Program
Box 134
Harvard Square
Cambridge, MA 02138
(617) 354-1216

□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□

NEW! CROMENCO Z-2!

The industry's most powerful microcomputer is only $595 in kit form. But it has a fast 250-nanosecond cycle time, the power of the Z-80 uP, and many special features. Cromemco, one of the best-known names in the microcomputer field, is introducing its new Z-2 microcomputer which is specifically designed to make available state-of-the-art power and speed in a low-cost microcomputer for dedicated work.

A significant feature of the new Z-2 is that it uses the Z-80 microprocessor, generally considered to be the standard of the next generation of microprocessors. The Z-80 has much more power and speed than such previous popular chips as the 8080, 6800, etc.

It's noteworthy, too, that Cromemco uses an especially fast version of the Z-80, one that has a speed of 4 MHz. Cromemco memories and peripherals are also designed to work at this speed, the fastest in the microcomputer field.

Essentials of the Z-2
The new Cromemco Z-2 is designed to provide the engineer or even the hobbyist with the industry's fastest and most powerful microcomputer in a form especially convenient to application in dedicated industrial, educational, instrumentation, laboratory, business and other work.

To give it maximum usefulness and convenience in these applications, the Z-2 contains (a) Cromemco's fast, powerful CPU card, (b) a motherboard with 21 card sockets, and (c) an extremely heavy duty power supply capable of meeting virtually any power need including floppy disk drives.

The above circuitry is contained in an all-metal housing arranged for either relay-rack or bench-cabinet mounting. The front panel of the computer is totally free and clear of controls or switches of any kind to make the computer immune to accidental program or memory mishaps caused by accidental switch flipping.

Heavy Duty Power Supply
The power supply in the Z-2 is designed to provide what can only be described as abundant power for all foreseeable combinations of circuits and peripherals including use of a floppy disk drive.

To this end, the Z-2 provides 30A from 8V, and 15A from both +18 and –18.

Broad Software Support
Since the Z-2 uses the Z-80 microprocessor, present 8080 software can generally be used with the Z-2. Cromemco, too, offers substantial software support including a monitor, assembler, and a control-BASIC interpreter which is specifically devised for microprocessor control applications.

S-100 Bus Gives Wide Peripheral Compatibility
The Z-2 uses the S-100 bus that has become standard in the microcomputer field. Dozens of manufacturers support this bus with compatible peripherals.

Cromemco itself has very well known compatible peripherals such as a 7-channel A/D *and* D/A converter; the low-cost BYTESAVER *which contains a PROM programmer* and provides capacity for 8K bytes of memory; a color graphics interface which permits an ordinary color TV set to be used as a *full-color* graphics terminal; a digital interface that provides two serial I/O ports, two 8bit parallel I/O ports and 10 independent programmable interval timers; and others. Cromenco also supports the Z-2 with fast 4 MHz dynamic RAM memories as mentioned earlier.

Rack or Cabinet Mounting
The Z-2 is supplied in a black-anodized metal case for mounting in a standard 19-inch relay rack. A quality stylized bench cabinet in an attractive blue color is also available.

Kit or Assembled
The Z-2 is available in either kit or assembled form. The kit comprises the Z-2 for rack mounting, the Cromemco 4 MHz microprocessor card, the full-length 21-slot motherboard, heavy duty power supply, one card socket and card-guide set, and front panel. The assembled Z-2 includes the above as well as all 21 sockets and card guides and a cooling fan.

Price/Delivery
Price of the Z-2 in kit form is only $595; delivery is from stock to 30 days.

For more information, contact:
Joe McCrate
Cromemco, Inc
2432 Charleston Rd
Mountain View, CA 94043
(415) 964-7400

□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□

MICROCOMPUTER SELECTRIC INTERFACING CARD

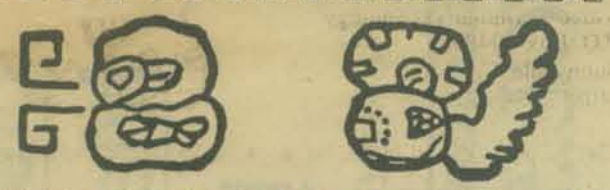
Interfacing a Selectric to your microcomputer? The Center for the Study of the Future, a religious information networking organization, has just released a Selectric interface card for any of the Dura or Itel Selectric terminals with 24 volt solenoids (Dura 1021, 1040, etc.) as well as for the Tycom adaptor. Input is RS-232 seven bit ASCII at 110 or 134 baud.

Application areas include word processing, mailing labels, stencil preparation, newsletter editing and composing, correspondence, information retrieval.....

The card is available in a kit form without power supplies for only $325, but is also available with power supplies as well as assembled. Some Dura units are also available as completed systems. And $15.00 will buy you a do-it-yourself book, including schematics, called *Interfacing Selectrics to Microcomputers.* For a brochure, send a SASE (stamped self-addressed envelope) to:

Center for the Study of the Future
4110 N.E. Alameda
Portland, OR 97212

□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□



□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□

ADD A CALCULATOR CHIP TO YOUR MICRO!

Now you can have full floating point calculations without using a lot of memory. With the RSG Electronics mathematical function unit printed circuit board, you get not only the standard add, subtract, multiply and divide, but a full 40 function floating point calculator array. Unique features include:

- Complete compatibility with any computer having one TTL level I/O port.
- Straightforward programming using I/O commands.
- Speed approximating and often better than software equivalents.
- Special functions may be programmed to operate independent of further processor control.
- Built-in overflow and error detection.
- A high quality board with through hole plating and gold plated fingers together with clear and complete instructions.

Is in stock now and available from:
RSG Electronics
P.O. Box 13
Santa Margarita, CA 93453

PC Board only	$24.95
PC Board plus CMOS chips	59.95
Postage and handling	1.23

California residents, please add 6% sales tax.

□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□■□

people's computer company

☆ $1 ☆

Z·80 Pilot

6502 Assembly Programming

Tiny Basic for Beginners

Free Software?

High School Computers

Math Drills & Games

FORTRAN Man

Reviews · Announcements · Letters

VOL.5 NO.5 MAR-APR 77

PCC
P.O. Box E
Menlo Park Ca.
94025

To: