PPX EXCHANGE

Vol. 1 Number 4 Copyright 1977 July 1977

A BIG HOWDY from Lubbock, Texas, the new home of the Texas Instruments Consumer Products Division. The Lubbock facility is the production and repair site for all TI consumer products. By relocating to Lubbock, PPX and the Calculator Staff will be better able to serve you. We apologize for any delay in correspondence, program submissions, orders, etc., caused by this move. Now it's business as usual and we look forward to offering you the best of service. Please refer **all** PPX correspondence to:

TEXAS INSTRUMENTS PPX
P. O. Box 53
Lubbock, TX 79408

PPX POTPOURRI

1. With the production of the TI-59, Texas Instruments announced the official opening of PPX-59, a software exchange for owners of TI Programmable 59's. Details indicate that PPX-59 will operate very much like PPX-52 has in the past, with similar pricing and exchange policies and a common newsletter, the PPX Exchange. PPX-52 and PPX-59 will operate independently of one another with separate catalogs, members guides, order forms, etc. Membership in one Exchange will not entitle one to membership privileges of the other.

2. In July, PPX-52 will release its first Software Catalog Addendum. From this point on, an Addendum will be issued whenever there are sufficient new submissions to warrant one. The Addendum will include not only the latest programs available from PPX, but an entirely new Key Word Index and Author Index. When necessary, a complete Software Catalog will be printed that will combine all the preceding addenda and catalogs.

PROGRAMMABLES "GO NAVY"

H. Alan Burkett, LCDR, Civil Engineer Corps, USN

In June 1977, the Naval Postgraduate School (NPS), Monterey, California, released my thesis (co-authored by LCDR Harry Kruse) entitled: "Investigation of Card Programmable and Chip Programmable Pocket Calculators and Calculator Systems For Use at Naval Postgraduate School and in the Naval Establishment." Research conducted from September 1976 through March 1977 focused primarily upon the SR-52 and HP67/97 systems as compared to each other, and to the projected TI-59 and National Semiconductor (NS-7100) systems. Among the parameters compared were calculator functions, basic and advanced programming techniques, and programmability, with emphasis on educational and practical applications. The thesis concluded that these machines provide significant advantages in teaching and learning mathematical concepts and that programmable pocket calculator systems are a potentially important management and tactical support tool, Navy-wide.

Concurrent with our research, NPS initiated several pilot projects using the SR-52 calculator. One such project involved a class conducted for 15 students in the Naval Intelligence curriculum. The course material included numerical procedures, Fourier analysis, differential equations, and the Laplace transformation. Although two weeks of class time was devoted to learning SR-52 capabilities and programming, this class covered 15% more material and completed a final examination judged to be 20% more difficult than any previous examination in similar courses. The overall grade point of the class was equivalent to that of the previous class! The following quarter these same students completed a graduate course which included solving differential equations with the Runge-Kutta method and computing cumulative/inverse, cumulative normal, and binomial distribution values, using programmable calculators, rather than referring to tables. Manual methods for solving these problems are normally too tedious and time consuming to be demonstrated past the "exposure" level; thus, the programmable calculators proved to be very advantageous to this class. (In this type of matrix and array manipulation, we found that the SR-52 had much more storage capacity than the HP67/97, especially when sacrificing program steps to gain extra data registers.)

Another interesting result was the "discovery" of the process we named "Thinking Process Transmutation," in which the programmable calculator user subtly, but inescapably, reorganizes his own thinking processes to fit the logical processes used by the calculator. Whether or not the user recognizes it, he acquires new capabilities to organize his thoughts, to define the necessary steps and then to develop the most efficient procedures to solve any problem. It is our strong belief that the refinements and complexities developed through this thought process carry over into all other portions of the individual's life. In short, after being told **what** to think all of one's life, programming calculators teaches one **how** to think!

As a follow-up to our thesis research, we have been asked to perform continuing analyses to determine the advantages associated with daily use of card-programmable calculators' in a Fleet environment. These analyses include daily tasks and special data manipulations in a variety of work situations. Measurements of programmable calculator's impact on reliability, maintainability, and operational readiness are being conducted in areas such as management and accounting, material and quality control, and engineering design and analysis. Standard programs are being developed where applicable and the feasibility of implementing Navy-wide use of these programs is being evaluated. In this effort, we are visiting various Naval commands to explain and demonstrate the advantages of using programmable calculators.

LCDR Kruse and I serve as points of contact, program reviewers, and as data collection/program standardization focal points for the Navy at 1220 Pacific Highway, San Diego, CA. 92132.

CALCULATOR DOCTOR

This column is intended to answer frequently occurring questions relating to either SR-52 operation or programming. These questions are obtained from TI's Consumer Relations Department. If you are having difficulty with your calculator or with programming, please contact TI's Consumer Relations Department for assistance.

QUESTION: I recently purchased an SR-52 and the optional Statistics Library. All of the programs run successfully, with the exception of the Random Number Generator program. Both the Basic Library (BA1-13) and the Statistics Library (ST1-04) programs give five random numbers and then start repeating these "random numbers." The cards are programmed correctly according to the listing in the books. What is wrong?

ANSWER: The Random Number Generator program was designed around the SR-52 as it was first produced. Unfortunately, the program will not work with later versions of the calculator, due to a slight change in the operating characteristics. For a corrected Random Number Generator program, please contact the TI Consumer Relations Department, Lubbock, Texas.

QUESTION: My SR-52, (in which I had extra memory capability installed by a non-TI source) lost a display segment four months after purchase. I sent it into the Service Facility in Lubbock to have it repaired. I sent it insured and enclosed a copy of my proof-of-purchase to verify the in-warranty status. I also enclosed a note describing the difficulty. I promptly received my calculator back from TI, COD, but it evidently was not repaired since the display was still defective. Why wasn't my calculator fixed, and why was it returned to me COD?

ANSWER: Texas Instruments does not support modification of the SR-52 to increase the number of available memories. In addition, the modification of your calculator by "a non-TI Source" automatically voided the 1-year warranty. The service facility is required to immediately return any modified calculator (regardless of the nature of the modification) to the sender in the same condition as it was received. The only way the Service Facility would be able to repair your calculator would be if you were to enclose a signed statement authorizing them to restore your calculator to as-manufactured condition (remove all modifications) as part of the repair process. You would, however, still be charged out-of-warranty repair charges, with your calculator being returned to you COD.

QUESTION: My calculator display is becoming difficult to read due to minor scratches on the lens. Is there any way I can correct this situation, without having to have a new lens installed?

ANSWER: Depending on the extent and depth of the scratches, you might be able to restore the lens to useable condition by using ordinary toothpaste as a rubbing compound. Clean the lens with a soft cloth moistened with water, then apply the toothpaste in a circular motion with a cotton ball. Change the cotton ball frequently and continue to apply the toothpaste until the surface scratches are gone. Remove any excess paste with a water moistened cotton ball.

THE JOY OF PC-100

Maurice E. T. Swinnen

We all know that the PC-100 print-cradle is capable of printing inputs/outputs, list entire programs, and trace program execution. The latter feature is the one that sold me. My first programs were developed and edited by single stepping back and forth to get the bugs out. Now it is a cinch and I am developing a pioneer spirit: "Remember the good, old days when we had to single step?"

You might not know that the use of the PC-100 greatly reduces our dependence on the user-defined keys. Before when using the SR-52 alone, we needed one user-defined key for each input/output, or we had to resort to the sometimes inconvenient use of the RUN key. We can now design programs in which pressing a single key produces a string of printouts. Add to this some strategically placed paper advance instructions and we get our data in presentable, easy to read blocks.

We can also modify existing programs, adding *prt and *pap instructions where needed. But, unfortunately, some of those programs are intended for SR-52 use only, and when print commands are inserted, they give double and even triple printings of the same value. This is especially true of print commands that use the same label as a subroutine in several segments. The trick to success here is to dump the result of the label into a memory register the first time it is used. If the label is used in a subsequent segment, delete it and insert a register call instead. Granted, you give up the convenience of being able to call the segments in any order, but you gain a neat printout. Consider the following example: *LBL, A, RCL, O, 1 +, RCL, O, 2, =, *prt, *rtn, *LBL, B, A, * √x, *prt, *rtn, *LBL, C, A, *log, *prt, *rtn. When A, B, and C are pressed in that order, the printouts will be A, A, B, A, C. Obviously we don't want A to print out that many times. The interpretation of the results becomes confusing and the margin of error increases considerably. So we rewrite: *LBL, A, RCL, O, 1, +, RCL, O, 2, =, STO, 1, 9, *prt, *rtn, *LBL, B, RCL, 1, 9, * √x, *prt, *rtn, *LBL, C, RCL, 1, 9, *log, *prt, *rtn. This works fine but is wasteful on user-defined keys. The real PC-100 pro writes: *LBL, A, RCL, O, 1, +, RCL, O, 2, =, *prt, * √x, *prt, *x^2, *log, *prt, *rtn. With 17 steps instead of 31, you get a printout of all three results, each one printed just once, always in the same, easy to remember sequence and by pressing just one key.

The PC-100 is usually used for printing inputs/outputs. But the printer can do much more. The first one to capitalize on this idea was R. Carlisle Philips with his PC-100 Plotter program (PPX #900001). He used the decimal point to plot the independent variable, using for contrast eights and ones on either side of the curve. A somewhat similar program was published by Warren B. Offutt in **Electronics**, March 1977. He also used the decimal point for plotting, but used only ones for background. Although very practical, both programs produce a printout suffering from a certain lack of contrast or visibility of the curve. It is possible to improve upon this visibility by printing a string of digits below each point on the curve, but only one zero above the point. An analysis of Warren B. Offutt's routine shows in essence the following: *LBL, A, STO, O, O, 9, *1/x, *LBL, sin, INV, *dsz, *1', X, 1, 0, =, GTO, sin, *LBL, *1', *prt, HLT. Enter a number between 1 and 11 and press A. The decimal point moves up the y-axis if the number is low, down if the number is high. Replacing 9, *1/x, (in the above) by 4,.,5, *1/x, gives a background of twos. You can experiment with 3, 2.25, 1.8, 1.5 and 1.125 for other backgrounds; however, contrast does not seem to improve noticeably. Try this short program segment: *LBL, A, STO, 0, 0, ., 8, *LBL, sin, INV, *dsz, *1', ÷, 1, 0, +, ., 8, =, GTO, sin, *LBL, *1', *prt, HLT. Enter a number between 1 and 10 and press A. Now a high number moves the decimal point up, a low one moves it down. Below the curve we have a field of eights but above the curve we only have one single zero per point. The background field can easily be changed by replacing both eights in the program segment by any other pair of digits, identical ones or different ones.

Another eye-pleasing plot is the bar graph, (PPX #900013) which can be obtained by the following sequence: *LBL, A, *CMs, +, 1, =, STO, 0, 0, *LBL, sin, INV, *dsz, *1', RCL, 0, 1, X, 1, 0, +, 1, =, STO, 0, 1, GTO, sin, *LBL, *1', *prt, HLT. Enter a number from 1 to 10 and press A. The printer will now produce a string of ones of length corresponding to the number entered. An interesting variation to the bar graph plot is the following: *LBL, A, STO, 0, 0, *LBL, sin, INV, *dsz, *1', X, 1, O, +, 1, =, GTO, sin, *LBL, *1', *prt, HLT. Enter a number between 0 and 9 and press A. The same type of bar graph results, but the first digit of the string is not a 1 but the number entered, indicating the relative height of the bar.

Another interesting way to show off your PC-100 is with matrix printing. For example, key in the following: *LBL, A, *fix, 3, *pap, RCL, 0, 1, *A', RCL, 0, 2, *B', RCL, 0, 3, *C', RCL, 0, 4, *A', RCL, 0, 5, *B', RCL, 0, 6, *C', RCL, 0, 7, *A', RCL, 0, 8, *B', RCL, 0, 9, *C', INV, *fix, *rtn, *LBL, *A', ÷, 1, 0, =, +, *rtn, *LBL, *B', ÷, 1, 0, 0, =, +, *rtn, *LBL, *C', ÷, 3, INV, *log, =, *prt, *rtn. This routine will print out all nine data memory contents, provided you have stored only one digit in each register. To try it out, store 1 in RO1, 2 in RO2 up to 9 in R09, and press A. Because of the inability of the PC-100 to print leading zeroes, each horizontal row of the matrix is presented as a fix 3 decimal. (Merely ignore the zeroes to the left of the decimal points.) This routine is perfect as a read-out for tic-tac-toe: zeroes for the unoccupied squares, ones for the player, eights for the machine. (B. R. Kelso's Hexpawn (PPX #910012) would be a perfect candidate for this routine.) In games like these it is always more realistic if the printer keeps score, not the player. The SR-52 has enough memory available between data and parenthesis registers to support even a 5 by 6 matrix.

Even though I develop a lot of game programs, I also use my PC-100 for some very down to earth purposes. The satisfaction I gain here is different; it makes my life easier by saving time and effort and by reducing the number of mistakes to a minimum. For example, since I began reconciling my checkbook using the Reconcile Checking Account program (BA1-07), I have had fewer problems balancing my account. However, because I didn't have a printout, my checking account still caused me problems. To solve this problem, make the following changes to BA1-07: 1) After *LBL D insert two steps *iferr, sin
2) After *LBL E insert two steps; *iferr cos
3) After the last step of the program, which with the 4 insertions is now 133, add: HLT, STO, 1, 0, +/−, *prt, RCL, 0, 8, -, RCL, 0, 9, -, RCL, 1, 0, = STO, O, 8, *prt, GTO, 1, 3, 4, *LBL, sin, SUM, O 8, *pap, *prt, RCL, 0, 8, *prt, *pap, GTO, 1, 3, 4, *LBL, cos, *CMs, STO, 0, 9, *fix, 2, *pap, *prt, *pap, GTO, 1, 3, 4.

Now when I write out checks to pay my bills at the end of the month (that's when I make most of my mistakes) I install the SR-52/PC-100 on my desk and proceed as follows: My bank charges 10 cents per check, so I enter this as ., 1, INV, E. If your bank does not charge you anything enter this as 0, INV, E. Then I enter my current checkbook balance and press INV, D. Also any deposit, such as the monthly pay check, is entered as INV, D. Now I enter the amount of each check (expense) as a **positive** number and I press RUN after each amount. The printer shows each amount as a **negative** number, subtracts it from the balance, subtracts the charge per check and prints out the new balance. I can enter deposits at any time, without fear of getting the program out of step. All deposits are offset by spaces to make their identification easier when I transcribe the results into my checkbook. With a little ingenuity, variations in bank service charges can be written into this program.

By using my PC100, I no longer have to depend on user-defined keys or the run key. I can also list strings of outputs as well as print answers individually. Pictorial printing, in the form of bar graphs and banners, is within my grasp. This powerful tool has removed the drudgery from my programming, and has actually put the "joy" back in.

FROM THE ANALYST'S DESK

• If you have purchased the "Yahtzee" program (PPX #310013) and have noticed that the last die is always a zero, we suggest the following modifications to the program:

As Listed:			Change to:		
000	99	*pap	000	34	tan
001	33	cos	001	03	3
002	06	6	002	69	*9'
003	06	6	003	69	*9'
004	06	6	004	69	*9'
005	06	6	005	69	*9'
006	93	.	006	79	*6'
007	99	*pap	007	77	*4'
144	05	5	144	06	6
153	03	3	153	04	4
162	01	1	162	02	2
170	08	8	170	09	9
171	09	9	171	00	0
180	07	7	180	08	8
187	65	X	187	55	÷
188	42	STO	188	44	SUM

With these changes, the program runs as originally documented, except that an additional trailing zero is displayed. This zero should be ignored.

• If you have purchased the "Degrees, Minutes and Seconds Display" program (PPX #900037) and have noticed that the seconds are always displayed in units of ten (10, 20, 30, etc.) then we suggest the following modifications to the program:

As Listed:			Change To:		
079	06	6	079	05	5
112	00	0	112	34	tan
113	27	*INV	113	93	.
114	49	*PROD	114	99	*pap
115	99	*pap	115	94	+/−
116	95	=	116	58	*dsz
117	99	*pap	117	99	*pap
118	77	*4	118	09	9
119	77	*4	119	10	*E'

With these changes the program runs as originally documented, but the last digit should be ignored.

• Since the original mention of fractured displays (May, PPX Exchange) many members have requested an in-depth discussion on how to generate these displays. They also noted that not all SR-52's create them in the same way. To clarify this unique feature of the SR-52, we offer the following detailed discussion of "Fractured Displays."

A fractured display occurs when the = key is pressed, after a so-called "mask" has been formed in the first pending operation register (register 60). A mask is a string of codes, which, by means of a special modification, causes various non-standard symbols to appear in the display. As was noted in the last newsletter, a mask is created by loading a string of mask codes into the display register, performing an arithmetic function ($\sqrt[x]{y}$, y^x, ÷, X, -, or

+) to push the codes into register 60, and then modifying the codes by a display-to-memory operation (STO, *EXC, SUM, *PROD, INV SUM, or INV *PROD) on register 60. There are thirty-six different combinations of arithmetic operations/display-to-memory functions, many of which produce different results. For simplicity, all further discussion will be limited to the combination . . . ÷ SUM 60 . . . =, with the mantissa of the mask positive and the exponent negative. For this particular combination, the mask codes are as follows: a 5 will produce a degree sign, a 4 will produce a minute sign (single quote), a 2 will produce a second sign (double quote), a 6 will produce a minus sign (hyphen), a 3 or a 7 will produce a blank, and 0, 1, 8, and 9 are transparent. In addition, the codes 8 and 9 will permit leading zero's.

Upon pressing the = key, two things happen. First, the number currently in the display register is modified (fractured) according to the mask in register 60. Second, both the mantissa of the mask and the mantissa of the number currently in the display register are shifted three digits to the left. Although the SR-52 displays 10 mantissa digits, extra digits which are normally not displayed are maintained for accuracy. The number of extra digits, known as "guard" digits, varies from 2 to 3, depending on when the particular calculator was manufactured. Normally, this variation is not significant, since only 10 digits are displayed, but because a fractured display causes a shift of 3 digits to the left, those calculators with only 2 guard digits will always show a zero as the rightmost displayed mantissa digit (after the shift). Regardless of the number of guard digits maintained, this digit will never fracture. (This non-suppressable zero has caused some consternation among users of the "Yahtzee" and "Degrees, Minutes, and Seconds Display" programs (PPX #910013 and #900037). The appropriate revisions to these programs are provided above.) The simplest way to tell which type of calculator you have is to key the following sequence: 3, *1/x, -, 3, *1/x, =. If your displayed answer is zero, then your calculator has 2 guard digits.

The following describes how to develop a mask which causes the calculator to simulate a LRN mode display. (Do not key in the following; merely follow the building blocks as we construct our mask.) The end result should appear:

xxx xx

where the x's represent numbers. The unique feature of this display is a space between the third and fourth numbers. Leading zeros are desired, so we will use 9's for the mask codes:

999 99

We wish for everything else to be blank, and therefore will fill the remaining positions with 3's (including the exponent):

3333999399-33

We must use a negative exponent according to the conventions mentioned above. Since the mantissa will be shifted three digits to the left, we need to insure that the last three mantissa digits are in the guard digits. To do this, we will need to add three arbitrary digits on the left end of the mantissa:

1.003333999399-33

Note the placement of the decimal point, which insures that the exponent will not change. Our building blocks are now complete.

Since the mask we developed above requires a thirteen digit mantissa, it cannot be keyed in directly. This number can be developed by storing it in a register and then summing in the necessary guard digits. There are two ways to do this; from the keyboard or under program control.

From the keyboard, if using register 01, key in the following: 1.003333999 EE 33 +/-STO 01 3.99 EE 43 +/- SUM 01. The next process is to choose the numbers you wish to be displayed. For this example, we will attempt to display 979 55. First, we must arrange each number to match our mask codes: 1.000000979055. Again we have a thirteen digit mantissa, obtainable by the use of a storage register, as mentioned above. Therefore, key in: 1.000000979 STO 02 5.5 EE 11 +/- SUM 02. Now, to create a fractured display from the keyboard, press: RCL 01 ÷ SUM 60 RCL 02 =. Your display should now appear 979 55.

To implement the above example under program control key in the following program sequence:

```
*rset LRN *LBL A STO 02 1.003333999
EE 33 +/-STO 01 3.99 EE 43 +/-
SUM 01 0 HLT STO 03
*LBL B 1 EE 22 STO 04 CLR
RCL 02 EE 13 SUM 04 CLR
RCL 03 EE 10 SUM 04
RCL 01 ÷ SUM 60 RCL 04 HLT LRN
```

Key in a one, two, or three digit number (for our example, 979) and press A. Then key in a one or two digit number (for our example, 55) and press RUN. When a number appears in the display, press =. To regenerate the fractured display without having to key the numbers in again, just press CLR, B, then press =. The exponent 22 at step 043 was an arbitrary choice - any other exponent would work just as well, as long as the exponents at 053 and 063 were changed accordingly. For example, if we had chosen an exponent of 23 at location 043, instead of 22, then the exponents at 053 and 063 would have had to be 14 and 11, respectively.

If your calculator has two guard digits, you will find that the last digit of the fractured display produced by the simulated LRN mode program listed above will always be zero. It may prove instructive, even to those whose calculators have three guard digits, to try to modify the program to shift both the mask and the number to be fractured one digit to the left. We will print this modification in the next issue of PPX-Exchange.

With careful study of the above example and some practice, one should be able to create fractured displays of all types, adding a new dimension to programming. Although fractured displays are somewhat complex to implement, they can be useful, and almost as important, enjoyable.

EDITOR'S NOTE: Due to the nature of fractured displays, the results of your efforts may be - NOTHING (especially if you use a negative mantissa)! The only way to really understand "fractured" displays is to experiment with them.

The PPX Exchange is published every other month and is the only newsletter published by Texas Instruments for SR-52 owners. You are invited to submit items you feel are of general interest to other SR-52 users. Inputs should be limited to 3 double-spaced typed pages. Please forward your newsletter inputs and any questions to:
TEXAS INSTRUMENTS PPX
P. O. Box 53
Lubbock, TX 79408