52-NOTES

Volume 2 Number 1 48/39 January 1977

Newsletter of the SR-52 Users Club
published at
9459 Taylorsville Road
Dayton, OH 45424

PUZZLING SEQUENCES

It now appears that the topic "Pseudo Behavior" (see V1N5p3) is not big enough to catch all of what is being discovered in the way of strange behavior, so I will consider in this space reported combinations of anything manually, SST, *bst, or program executable that produces a new result. I'll explain what I can, and invite corrections, enhancements, new explanations, etc from the membership at large. Henceforth, pseudos will be designated with a p followed by the synthetically creatable 2-digit code (p31 means pseudo 31); keyboard-creatable instructions will be designated by the appropriate mnemonic (*INV' is op-code 27).

A *LBL Peculiarity (52): Michael Brown (128) and Dallas Egbert (384) have found that if in RUN mode keying LRN is preceded by *LBL, the program step pointer is advanced one step. Expanding this a little further, I find that *LBL SST LRN advances the pointer 2 steps. It also appears that the pointer moves one step if the *LBL and the LRN are separated by keyed numerals (which leave a soft display). However, *LBL CLR LRN does not move the pointer. Then, if the pointer is positioned at step 223 with *LBL LRN executed manually in RUN MODE, apparently nothing happens (a second LRN reveals step 223 in LRN mode). This is probably related to what happens when you execute an instruction at step 223 with SST: it takes 2 successive LRNs to get into LRN mode. Incidently, *LBL p31 appears to behave like any other label under program control.

*INV' Acting Like p21: Dallas has been experimenting with sequences involving *INV' that make it appear to behave like the p21 pseudo (2nd). It should be noted that while p21 as a separate step doesn't always behave like a "2nd" when merged with an instruction which it shifts, in the cases Dallas tried, it does: put *INV' at step 223 (99 for SR-56), then in RUN mode (at step 223 (99)) key SST, and it will take 3 successive LRNs to get a switch to LRN mode; write beginning at step 000: *LBL A *INV' p31, press A, press any non-edit button, then backstep to find the code for the shifted operation of the previously keyed button; press A (in RUN mode), then SST, and find that it has executed as *bst. Substitute p21 for *INV' in the above sequences, and the results should be the same. This observed behavior suggests that program or SST execution of either *INV' or p21 can produce a viable detached pending shift which can affect manually keyed operations in either RUN or LRN modes. In the 3 LRNs example, the first LRN is executed as *IND (*f(n)

The SR-52 Users Club is a non-profit loosely organized group of SR-52/56 owners/users who wish to get more out of their machines by exchanging ideas. Activity centers on a monthly newsletter, 52-NOTES edited and published by Richard C Vanderburgh in Dayton, Ohio. The SR-52 Users Club is neither sponsored nor officially sanctioned by Texas Instruments, Incorporated. Membership is open to any interested person; suggested contributions are $6.00 per six issues ($10.00 abroad).

for SR-56) and it takes 2 more following a step 223 (99) SST to get into LRN mode (see "A *LBL Peculiarity" above). Speaking of label peculiarities, unlike *INV', p21 when addressed as a label causes a reset with error condition.

SST'd SST (52): In the course of his examining *INV' behavior, Dallas found that execution of p71 (SST) by a manually keyed SST causes the program pointer to skip over (or perhaps execute) the p71 and to execute the next instruction. This effect appears to hold for any number of successive p71s. (In RUN mode key: 1.009717171 STO 70 GTO 003 SST, and see the trailing 9 that was executed after the program pointer skipped over (executed?) the 3 p71s)

.001 Dim Display: Chuck Sanford (214) finds that with the SR-52 plugged into the PC-100, the .0000001 dimming (V1N3p6) does not occur, but that .001 causes dimming. Thus there appears to be a power supply dependency for this display phenomenon (which holds for the SR-56 also).

ALPHABETIZED MEMBERSHIP LIST

Several members with access to general purpose computer systems have offered to prepare alphabetized membership lists, and I greatly appreciate the volunteered help. The most attractive came from Michael Brown (128) who managed to get an IBM 1130 system to type the output directly on to mimeo masters. The resulting list is included with this issue of 52-NOTES that goes to members. Many thanks Mike for providing a valuable service that minimizes the publishing effort. Incidently, I will note here that I intend to continue printing 52-NOTES via my home-based mimeograph until such time as circulation should become too large to handle. I like to be able to get ephemeral topics in on short notice, and will assume that a majority of the membership prefers topical timeliness to unblemished printing (until I get word to the contrary).

TI NOTES

Machine Anomalies: To date, TI has issued one "addendum" for the SR-56; none for the SR-52. The one for the SR-56 was reportedly included with the first retailed unit, but apparently Sandy Greenfarb (200) didn't get one. The jist is: m + or - 0 y^x n=mD12 and m + or - xrty n = mD11 (the xrty part courtesy of Sandy, who discovered both effects while debugging a practical program).

Games PAC: The Games bonus offered in the December Scientific American TI advertisement to buyers of new SR-52s will be available as a regular PAC 15 January 1977.

PPX-52 Publications: The first issue of the PPX-52 Newsletter was due to be published the last week of December. The next catalog is due about February.

Machine Exchange: TI now has a number of centers at which users may exchange defective machines (in warranty) for new ones. Call Consumer Relations for the one nearest you.

CORRECTIONS

SR-52 Timer Instructions (V1N7p6): Step 8 should read: "For new timing exercise, go to step 2".

Routine B (V1N5p5): The second *fix 0 is superfluous.

REGISTER BEHAVIOR: Part III OVERFLOW and UNDERFLOW

Herewith the article that I mistakenly said last month would be in V1N7.

The SR-52 and SR-56 Owner's Manuals tell us that both machines were designed to be in an error state any time either display or register arithmetic causes (or tries to cause) the creation of a number whose absolute value is smaller than 1D-99 or larger than 9.999999999D99, and, for the SR-52 that "... When direct register arithmetic results in underflow or overflow of that register, the error condition remains until the contents of that register are changed." (see page 183 of the SR-52 manual). As it turns out, it is possible to create a number larger than 9.999999999D99 without triggering an error state, and the presence of an error-state producing overflow or underflow value in a register does not of itself cause (or continue to cause) an error state. If the 11th, 12th, and 13th places are 499 or less, then there is no overflow, and if register arithmetic causes an overflow, CE executed either manually or under program control appears to turn off the error state. A subsequent recall of an out-of-bounds number would, ofcourse, re-invoke the error state. And then, it's even possible for an SR-52 program-memory register to contain a sequence of op-codes which if interpreted as data would constitute an overflow or underflow value. For example, in RUN mode key: 9.999999999 EE 99 STO 70 9.99 EE 89 SUM 70. This should not cause the machine to go into an error state. Yet, if RCL 70 is keyed, an overflow error condition is created. Or, if (SR-52 only) the sequence: +/- *PROD 0 0 0 0 0 0 is written starting at step 000, then RCL 70 creates an underflow error condition.

In the discussion that follows, let's examine register contents as they appear in op-code format, using the designators: AB, CD, ... OP (see V1N1p5), determine the specific characteristics that cause overflow and underflow, and consider the concepts of "conditional" and "absolute" overflow. Key the sequence: 9.999999999 EE 99 STO 70 4.99 EE 89 SUM 70. Reg 70 should now contain the number 9.99999999499D99. Now RCL 70. All 9's should show, but without error condition. Now key: 1 EE 87 Sum 70. No error condition; but after RCL 70 there is. Steps 000-007 should look like: 90 09 50 99 99 99 99 99 which represent the number 9.99999999500D99. Note that a one was added to the 13th place changing positions CD from 99 to 09, and EF from 49 to 50. Register arithmetic occurred properly, i.e.: 9.99999999499D99 + 1D87 = 9.99999999500D99. Yet the resulting sum is regarded as too large when displayed, because of the attempted rounding to ten significant figures. I suggest that Reg 70 is now conditionally overflowed: it contains a number which can be operated upon normally by register arithmetic, but which when displayed creates an overflow error state. This is true for all numbers in the range from 9.999999999500D99 to 9.999999999999D99 inclusive. The setting of the overflow error state during register arithmetic only occurs when the result would be larger than 9.999999999999D99, in which case this conditional overflow number is left in the register.

Now let's see what happens with underflow. In RUN mode key: 1 EE +/- 99 STO 70. Steps 000-007 should look like: 94 09 00 00 00 00 00 10. This is the smallest number the machine can hold (putting anything but zeros in the 11th, 12th or 13th places (positions E, F, C) only makes the number larger). Now try to make the number smaller by keying: .1 *PROD 70. No error condition. Analogous to overflow, the number left in the register is the smallest possible, but unlike overflow, the register is not "conditionally" underflowed; i.e. bringing it into the

display does not cause an error condition. Now let's see what happens when we artificially create a "number" that is too small to recognize as a datum (SR-52 only). Cycle the on-off switch, then in LRN mode at step 001 key *E'. Now in RUN mode key RCL 70, and see 1D-12. The machine moved the one at position C 12 positions down to position 0 in order to format it acceptably for display (see V1N2p2) and compensated for this by the D-12. Now, in LRN mode beginning at step 000 key: +/- *D' 0 0 0 0 0 0. From what happened in the preceding example, we might expect the machine to interpret this "number" as 1D-111 which is, ofcourse, too small to display. Indeed, if in RUN mode you RCL 70, you will get a flashed 1D-99. The smallest artificially creatable legal number would look like: 84 09 00 00 00 00 00 01 which transfers as 1D-99. Apparently artificially created too-small numbers cannot be expanded via register arithmetic to become "legal". If 1D-100 (represented in steps 000-007 by 94 09 00 00 00 00 00 01) is multiplied by 10 (via register arithmetic), the result is 1D-98 (instead of the correct 1D-99) and an error state created. In fact, register arithmetic performed on any artificially created underflow number, treats that number as 1D-99 before operating on it.

Some of the implications of all this are: 1) Either conditionally or absolutely overflowed registers will respond normally to register arithmetic, provided absolute overflow does not result, 2) RCL of either conditionally or absolutely overflowed registers creates an error condition, 3) there is no conditional underflow, 4) RCL of an underflowed register does not create an error condition, and 5) register contents that are interpreted as less than 1D-99 are treated as 1D-99 during either register arithmetic or when recalled, and an error condition is created in either case.

THE MATRIX CHALLENGE (52)

Some of us have found it both challenging and rewarding to write programs that solve common matrix problems. Much of the challenge lies in how to determine the best approach, as well as how best to mechanize the chosen one. I first confronted the SR-52 4 X 4 determinant problem a year ago, and thought that a fairly straight forward FORTRAN algorithm for a Gaussian Elimination method would work well. I managed to get a working program to fit on one card (which could also be modified to get a 5 X 5 determinant) but it wouldn't handle column exchange when zeros were produced on the diagonal. So I modified the main program so that it would automatically read a second card that would exchange columns if required (4 X 4 only). It was great fun working out multiple pointer manipulations and program overlays, but the program didn't perform nearly as well as Dix Fulton's straight forward approach (V1N3p4). I haven't yet come across a program superior to Dix's for just calculating a 4 X 4 determinant, but it is tempting to try to squeeze in other matrix operations all on one card. Using a table lookup scheme to save space (V1N4p2) Alan Trimble tacked on a solution to 4 simultaneous equations that works well, but appears to be surpassed by Rick Wenger's (235) program (below) which runs faster and requires fewer steps, mechanized with an extensive network of subroutines. Incidently, there is a clever flag shortcut in Rick's program: a call to the undefined D function sets the "flag" and *iferr provides the test. (It's just too bad if a real error shows up!)

For a matrix inverse solution, it appears that no matter what approach you take, there are a lot of computations involved... more than can be fit into a one-card SR-52 program, without clever manipulations/

tricks. I tried a table lookup optimization (V1N5p4,5,6) approach, but still couldn't squeeze it all in. However, there is at least one way that works, as Barbara Osofsky (420) shows in the program that follows (modified with a few I/O, throughput, and space-saving enhancements, including Rick's flag shortcut). A 92-step subroutine does all the arithmetic; the other 132 steps handle input, output and the required register manipulation to make 20 calls to subroutine *A'. Although this program was written to be used with the printer, it can be used without the printer by substituting HLTs for the *prts at steps 082 and 191. Barbara claims to be close to a one-card 5 X 5 inversion... good luck! Then, ofcourse, it would be nice to combine the determinant, inverse, and equations solutions all on one card. The four equations constants can be input following the matrix elements in Barbara's program, but Cramer's Rule column exchanges would need to be done manually. Perhaps someone can combine Rick's and Barbara's programs into one!

SR-52 Program: 4 Simultaneous Equations — Rick Wenger (235)

User Instructions: Same as for the V1N4p2 program.

Program Listing

```
000: *LBL E STO 68 *IND RCL 68 *rtn *LBL *A' E X RCL 15 - RCL 14 X *rtn
023: *LBL *B' E ) X ( *rtn *LBL*C' E X *rtn *LBL *D' E - *rtn
040: *LBL *E' E ) + ( *rtn *LBL C E X RCL 13 - RCL 12 X *rtn
*LBL B *IND RCL 68 *IND *EXC 69 *LBL A *IND E 1 SUM 68 SUM 69 RCL 68 *rtn
085: 0 STO 69 D (10 *A' 11 *B' 0 *C' 5 *D' 1 *C' 4 *E' 6 *A' 7 *B' 1 *C'
111: 8 *D' 0 *C' 9 *E'2 *A' 3 *B' 4 *C' 9 *D' 5 *C' 8 *E' 8 C 9 *B'
133: 2 *C' 7 *D' 3 *C' 6 *E' 4 C 5 *B' 3 *C' 10 *D' 2 *C' 11 *E' 0 C
157: 1 *B' 6 *C' 11 *D' 7 *C' 10 *E' 0 = *iferr 197 div RCL 66 = HLT
181: 1 +/- *PROD 66 16 E B B B B GTO 090 STO 66 CE GTO 186
```

- - - - - - -

SR-52 Program: 4 X 4 Determinant and Inverse — Barbara Osofsky (420)/Ed

User Instructions

1. Initialize with CLR
2. Key first element, press E; see 1 displayed, and printed confirmation
3. Key ith element, press E or RUN; see i displayed and printed confirmation. Repeat for i = 2, 3, ..., 16 with row-wise catenation
4. Press A, and get printed the determinant, followed by the 16 inverse elements grouped by rows. Program ends with -1 displayed. For a new problrm, go to step 1.

Program Listing

```
000: *LBL *A' RCL 05 X (RCL 10 X RCL 15 - RCL 14 X RCL 11 ) + RCL 06 X
028: (RCL 11 X RCL 13 - RCL 15 X RCL 09 ) + RCL 07 X (RCL 09 X RCL 14 -
059: RCL 13 X RCL 10 = div RCL 99 +/- STO 99 +/- = *iffl g 0 083 *prt
083: X RCL 00 = SUM 69 *rtn *LBL C + 3 STO 98 = STO 68 *IND RCL 98
107: *IND *EXC 68 *IND STO 98 4 *iferr 121 1 +/- SUM 68 SUM 98 RCL 98
131: *ifpos 103 CE *rtn *LBL B *A' 4 C *A' 8 C *A' 12 C *A' 12 C 8 C
155: 4 C *rtn *LBL *D' *pap D + 1 +/- *PROD 99 12 GTO 096 *LBL A *pap
177: *stflg 0 1 STO 99 CLR B INV *stflg 0 RCL 69 *prt *pap STO 99 B 1
198: *D' B 2 *D' B 3 *D' B *rtn *LBL E *IND STO 69 *prt 1 SUM 69
218: RCL 69 HLT GTO E
```

- - - - - - - -

FORUM

PC-100 Hardware Problems: Gerald Donnelly (203) has had several printer problems, and wonders what is the best way to get satisfaction. Since it now appears that many of the early PC-100s were poorly designed or fabricated, my advice is to keep exchanging units until you get a good one (Gerald and I are already on our second replacements). Just make sure you find all the faults before your warranty expires! Gerald suggests that it might be helpful to the membership to tally incidences of hardware problems. Let me know yours, and I'll pass the info along in a future newsletter.

Duplicate PPX-52 Programs: Barbara Osofsky (420) brings up the matter of PPX-52 possibly rejecting a program because it produces the same results as one already in the library, even though it may be superior in some way. She suggests that Club members could share their PPX-52-rejected but superior programs. This looks like a good idea, and I invite members who wish to participate to send me an abstract of a rejected superior program, the ways it is superior to a corresponding PPX-52 program, and to be willing to provide copies to members who send a SASE and money (state how much you require) to cover copying costs. I will publish such declarations in future newsletters.

SR-52 Pause Function: Don Williams (29) and Shuichi Takahashi (422) would like to know if anyone has been able to create an SR-56-type pause function for the SR-52.

BOOK REVIEW: Applied Mathematical Physics with Programmable Pocket Calculators Robert M Eisberg (McGraw-Hill 1976, 176 pages)

The capabilities and availability of the SR-56 and HP-25 class of personal programmables led physics professor Eisberg to consider a new approach to introducing selected topics in physics at the college level. By mechanizing elementary numerical solutions to differentiation, integration, and differential equations for both the SR-56 and HP-25, Eisberg introduces physics problems to the novice which would be either too difficult or just plain unsolvable using analytic techniques. Topics range over linear and central-force motion with friction taken into account; mechanical and quantum theory oscillators with resonance, damping, and coupling; and random processes. The programs appear to work, although most could be readily optimized for more efficient I/O. And since most output is destined to be plotted, incorporation of the PC-100 printer with the SR-56 programs would be a significant enhancement. This looks like a good book for layman personal programmable users as well as physics instructors and students.

CORRECTION: V2N1p4 (bottom)

The sequence: 1 EE +/- 99 STO 70 .1 *PROD 70 does create an error condition; subsequent RCL 70 does not.

MEMBERSHIP ADDRESS CHANGE

Make the following change to your membership list: #281: #18 (not #17).