52-NOTES

Volume 1 Number 7 48/48 December 1976

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

SEASON'S GREETINGS TO ALL!

It has been very gratifying to receive so many letters that say you like the way the Club and 52-NOTES have been going. I'll do my best to continue to live up to your expectations, and take this opportunity to wish you and yours a happy holiday season and a challenging year ahead.

"ROUNDING" TERMINOLOGY

I have been guilty of using the term "round-up" rather loosely, and will herewith define specific terms that will apply to 52-NOTES usage henceforth. I will use the word "round" by itself to mean what is sometimes referred to as round-off, or symmetrical rounding, and is what I meant by using "round-up" in V1N1p2, V1N5p4 and V1N6p2: Digits 5 through 9 increment the next higher digit (place) by one; 0 through 4 have no effect. The term "round-up" will be used when all the digits 1-9 cause the next higher digit to be incremented by one. If I ever use "round-down", it will be synonymous with "truncate". Thus 1.2 rounds to 1, but rounds-up to 2; 9.8 both rounds and rounds up to 10, but truncates to 9.

THE FIRST TI PPX-52 CATALOG

TI has published and mailed to PPX-52 members the first Software Catalog (November 1976) which partitions SR-52 programs into 11 major categories, which are further subdivided into a total of 99 topics. 62 of these are represented in this first catalog, leaving 37 still to be addressed by contributors. Of some 450 programs, approximately 235 were contributed; 215 selected from TI's applications library. I expect that program quality will vary considerably from program to program, and there are already indications of functional duplication. As Club members who have joined PPX-52 come across outstanding programs, I invite them to describe them and their key features. I will pass such information along via 52-NOTES, but will not publish PPX-52 programs, per se.

BRINGING THE SR-56 INTO SR-52 DISCUSSIONS

I now have an SR-56, and beginning with this issue will attempt to include the SR-56 in SR-52 technical discussions wherever practicable. In order to minimize special-case explanations, I will adopt a few convenient conventions, which I will discuss here along with noting critical differences and similarities. 52-NOTES titles expected to be of interest to SR-52 users primarily will be followed by: (52); (56) for SR-56 users. The presence of neither indicates targeting to users of either machine. In general, SR-52 architecture will be the basis for discussions and routine listings when both machines are under consider-

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).

ation. Therefore I will devote some space here to an explanation of how SR-56 users can interpret SR-52ese.

Although the SR-56 does not appear to give the user direct access to pending operations registers (Reg 60-69 on the SR-52) or program memory (Reg 70-97), comparable registers must exist for it to function as it does. Thus for purposes of discussion, Reg 60-66 will identify the pending operations registers of both machines (reference to Reg 67-69 applies only to the SR-52, as the SR-56 holds 3 fewer pending operations). However, no attempt will be made to individually identify SR-56 program registers. SR-56 users should familiarize themselves with the SR-52 program-memory register format (V1N1p4,5) in order to follow register behavior discussions that apply to both machines. Incidently, an important point to keep in mind is that the architecture of most of the common functions appears to be identical. Although the SR-56 user can't examine data in an 8-step op-code format, he can determine the values corresponding to the 16 places, and with a few restrictions, set them too. Following the A to P coding outlined in V1N1p4, the normal RUN mode display shows positions A B D G H I J K L M N O P. The missing C E F digits represent the 13th, 11th and 12th mantissa places, respectively, and a little maneuvering will reveal these. For example, the sequence: 1 e^x puts 2.718281828459 into the display register, but only 2.718281828 shows. To see the 459: STO 1 2.71 INV SUM 1 EE RCL 1, and see 8.281828459D-3. (Incidently, INV lnx in SR-52ese corresponds to e^x for the SR-56). In general, to see the last 3 digits of a 13-digit number, display it in decapower notation, store it, subtract from it the first 3 digits (including the indicated decapower) by register arithmetic, then recall it. To synthesize a 13-digit number: key the first 10 digits in decapower notation, store the display, and add to it (via register arithmetic) the last 3 digits with a decapower 10 less than that used for the first 10 digits. For example, e to 13 places can be synthesized by: 2.718281828 STO 1 4.59 EE +/- 10 SUM 1. The restrictions on building up the 16 places in this manner are that position B can only be 0, 2, 4 or 6 and position O cannot be made zero.

Thus the SR-56 user can follow many discussions and even try out some examples that deal with data in op-code format. So far as I know, however, he will not be able to create pseudos (V1N5p3) or an underflowed number (see the article on Overflow and Underflow elsewhere in this issue). SR-56 users should translate SR-52 register operations on 2-digit operands into appropriate 1-digit ones.

FORUM (52)

David Brown (107) finds both the SR-52 and TI's statistics library too EE oriented (complicated manipulations of small amounts of data) to do him much good with large amounts of data that he would like to process for social science types of applications. Although there aren't any personal programmables yet with hundreds or thousands of storage registers, the SR-52 has the most, and there are types of statistical processing that can handle any number of data inputs, where inputs are used only once, and do not need to be saved. For applications requiring separate storage of all inputs, unused program-memory registers can help to extend data storage capacity. These can be made contiguous with data storage registers (for indirect addressing) by beginning with the first unused program register, and ending with data register 119 (same as 19).

John Barnes (157) brings up the matter of desired features for a successor to the SR-52. He would like to see the instruction set be a superset of the SR-52's, have all program-memory register contents fully transferable (see V1N2p2), and have expandable memory (via hardware).

TIPS

Use of Pseudo 73 (52): L Frank Stallings (389) notes that pseudo 73 used in place of *rset works well in loops with flags, since it executes an effective *rset without resetting flags.

Short Absolute Branch (52): Frank also notes that a conditional branch can be made to an absolute address in the range 000-009 with only 2 designators (instead of 3). For example, the execution of: *ifzro sin 5 ... branches to step 005 for a zero display, provided *LBL sin has not been defined, or continues on if the display is not zero. For both cases, an error condition is created. J Wentz (61) and C Belanger (254) have also noted this effect.

Label Execution Anomaly (52): Frank goes on to suggest a peculiar execution of labels, which I tend to suspect may indicate a problem in his machine. Execution of the sequence: *LBL A SBR sin HLT *LBL B SBR INV HLT *LBL sin *LBL INV SUM 69 *rtn by: CLR 1 A RCL 69 in run mode should produce 1. If this is followed by B, according to Frank, Reg 69 would be decremented to zero, but I find Reg 69 contains 2 (which it should, following normal label behavior). If Reg 69 ends up being zero, then somehow the *LBL preceding INV gets ignored, and the INV is treated as a function, not a label. Anyone else experiencing this result, let me know.

Electro/Mechanical Tips: Marshall Williams (227) suggests the use of Brilliantshine metal polish to remove desplay-cover scratches, and found that an uneven key-touch problem due to key-panel adhesive oozing could be remedied by removing excess adhesive from the panel and keys with Energine flammable type spot remover. Marshall has designed an adaptor that allows SR-52/56 operation from a 12v DC supply, the circuit diagram for which he has offered to share with members sending him a SASE. A current-limiting feature presumably precludes damage that might be caused by power fluctuations.

Writing Good Diagnostic Programs (52): For members attempting to write effective, efficient diagnostic programs (V1N4p4), Jared Weinberger (221) suggests that dynamic code modification (V1N2p3) and (V1N3p5) could be used to advantage.

Execution Time Variations With Formatting: Mike Marquis (205) notes that display formatting affects program execution speed. He finds EE with *INV *fix is fastest; INV EE with *fix 8 the slowest. Mike also notes that register arithmetic is slower than display arithmetic. However, there are indications that the degree to which display formatting affects execution speed depends on datum values and the operations performed. Dallas Egbert (384) notes that 1d-99 recalls from data registers noticeably slower following a CLR. This appears to result from the CLR putting the display in an INV EE format. For data so recalled, execution speed decreases as values depart from 1. The rate is greatest toward + or - 1D-99, and least toward + or - 1D99. I have observed this phenomenon in both the SR-52 and SR-56 and it probably deserves further exploration. A more complete understanding of execution speed behavior can be especially helpful in the design of long-running programs.

Revision to Automacic Fill of Reg 60-69 (V1N1p3) (52): Several members have noted that the STO in routine E may be deleted. (the sequence starting at 00: X (RST executed by RST R/S in RUN mode fills the SR-56's pending registers).

More_On CLR(52) Stephen Franklin (217), Dallas Egbert (384) and Orla Damkjer (393) have all noted that the number that finds its way into Reg 60 following the sequence: CLR pseudo 83 CE on a hardened display (see V1N6p3), is not always the same as the originally displayed number. What appears to happen is that digit B (see V1N1p4,5) of the original number is "cleared" (made zero). The effect is to make both the mantissa and decapower signs always positive (V1N1p5). Thus, only numbers greater than or equal to one remain unchanged. This looks like a good way to get the absolute value of any 13-digit number in this range.

More_On Timed Crash (52): Dallas also notes that if the number in Reg 00 in Graham Kendall's Timed Crash (V1N6p3) is 1D12 or greater, execution does not cause a timed crash, but an immediate branch to step 000 (with no apparent resets). John Allen (104) finds this dividing line to be 1D13. Any others? (On my machine 9.999999999999D11 crashes but 1D12 does not).

More_On INV Viability (52): Jared Weinberger (221) notes that the sequence: CLR INV 5 *PROD 00 will divide Reg 00 by 5, but that leaving off the CLR neutralizes the INV, and Reg 00 is multiplied by 5. Further, the sequence: 2 INV 5 *PROD 00 divides Reg 00 by 25. Thus if there is a program requirement for a subroutine to divide a register by a constant, and another to multiply the same register by the same constant, something along the lines of: *LBL A 0 *LBL B INV 123 *PROD 00 *rtn does the trick (*LBL A 0 works as well as *LBL A CLR, and does not wipe out Reg 60-69). A call to A performs division of Reg 00 by 123; to B, multiplication, provided a "soft" display is not passed to routine B (i.e. the call to B is not preceded by CLR or numerals). Or, if two different constants are required, and they can be formed by partitioning a single string of digits, something like: *LBL A 123 *LBL B INV 456 *PROD 00 *rtn will divide Reg 00 by 123456 with a call to A, and multiply Reg 00 by 456 with a call to B (from a hardened display).

Charger Connection: John Allen (104) finds that the connector-short problem (V1N5p6) can be avoided by plugging the charger into a turned-on calculator before applying power to the charger.

More_On The 0 div 0 Error State: Stephen Bepko (45) notes that the 0 div 0 error state (see V1N1 p2) is cancelled upon completion of an arithmetic operation. He cites the following example to show this: in RUN mode key: 0 div 0 = CE 1 SUM 01 + 2 SUM 02 + 3 SUM 03 + 4 SUM 04. Then Rcl 01, 01, 03, 04, and find that INV sum applied to Reg 01 and 02, and that SUM apllied to Reg 03 and 04. Dallas Egbert (384) adds CLR, sin, cos, tan *D,MS, *P/R, y^x, xrty and *ifflg to the list of possible error-condition cancellers.

MEMBERSHIP ADDRESS CHANGES/CORRECTIONS

Make the following changes in your membership lists: 8: Philip, Dudley Observatory Plaza 7 1202 Troy-Schenectady Rd Latham NY. 25: Wilkins. 45: MD (not MO). 55: 14 Robin Rd Monmouth Jct, NJ 08852. 63: 6302 (not 63). 81: GF (not EW). 176: Box 209 Oceanport, NJ 07757. 255: 3502 Mount View Ave #9 Schofield, WI 54476. 274: Computer Center (Code 0141) Naval Postgraduate School, Monterey, CA 93940. 277: 5107 Calle Asilo, Santa Barbara, CA 93111. 42: 14228 Jefferson Ave #A Hawthorne, CA 90250.

ONE MORE (the last?!) ON SHOOTING STARS

J A Walston (291) gets all the stars in 5 moves (vs Stephen Bepko's 13 as reported in V1N6p6), and Michael Brown (128) claims that there are 82 unique 11-move solutions to regular Shooting Stars.

ROUTINES

Pending Parenthesis Extractor: Jared Weinberger (221) has devised a routine that counts the current number of ('s without disturbing pending operations. While the routine itself may not find much application, it reveals a new aspect of pending operation/parenthesis behavior. The routine is: *LBLA STO 99 10 STO 98 *LBL B (1 INV SUM 98 INV *iferr B CE RCL 99 *rtn, and is run by pressing A. As Jared notes, only pending ('s are counted (the number of them is returned in Reg 98), not pending operations (which aren't always separated by parentheses). Thus 5 + 4 X 3 y^x 2 ((A shows two ('s, while 5 + 4 X 3 y^x 2 A shows none, even though there are 3 pending operations in each case. Apparently the machine counts ('s separately from the number of pending operations. If you key ten ('s either consecutively or scattered around other non- =, CLR or) operations, an error condition is set (SR-56 too), just as if all the pending registers had been filled. A similar routine works for the SR-56. Starting at step 00, key: 9 STO 1 (1 INV SUM 1 GTO 03. Run with *CMs RST R/S. Execution halts with an error condition, and the number displayed prior to execution is lost. The number of current ('s is in Reg 1. The SR-56 appears to use the same parenthesis counter as does the SR-52, even though it has 3 fewer pending operations registers. As these routines suggest, ('s can serve as a loop control counter, thus freeing a register for other use.

A Partial Wipeout (52): John Allen (104) notes that if the sequence *LBL A *ifflg 0 A *LBL B is executed with flag 0 set, by a repeat of: A HLT B (until an unflashed display results), all registers are cleared, but the program pointer is positioned at the 3rd step ("0"), and the flag remains set. Jared Weinberger (221) notes the same results for ...INV *ifflg 0 ... with flag 0 unset. I find that these routines do not cause the partial wipeout for flags other than 0.

*INV' Crash: Al Roussin (64) notes that program execution of the sequence: *LBL A X *INV' = HLT causes a crash. But if X *INV' = is keyed manually, there is no crash, and if this is followed by another =, an error condition is created. For the SR-56, program execution results not in a crash, but an error condition, and manual execution squares the display after the second =. Incidently, although Al refers to *INV' as pseudo 27, it is LRN-mode creatable (code 17 on an SR-56).

More On Last Digits Viewers (52): The Parsons/Weinberger routine (V1N6p3,6) appears to work only for numbers greater than or equal to 1. (both signs must be positive). The Kendall routine seems to work for any number, and can be automated (doesn't need the manual =) as follows: starting at step 209: *LBL E STO 99 0 + STO 60 RCL 99 = pseudo 31. John Allen (104) reports that his machine (#015444) displays 4th through 13th digits with decapower zeros suppressed. Anyone else?

VALID COMPARISON OF TWO NUMBERS

Even a fair understanding of register behavior (V1N1p4 and V1N2p1,2) may not be sufficient to keep from making false assumptions that lead to incorrect number comparisons. An important potential hazzard is the assumption that display subtraction of one number from an identical one always produces zero. The large number of instances when the results are zero can be misleading, and the few that aren't can cause serious problems, especially when a zero test is made on a result that is expected to be zero but isn't. So long as the 13th place is zero, subtraction produces zero; otherwise a residual results (see V1N1p4 and V1N2p1). In general, any calculation that results in a decimal approximation has the potential of producing a non-zero 13th place. This includes the trig, log and exponential functions, 1/x, and x! for x GT 18 (*x! on 18 produces an exact result with a 13th place value of 8). Both display and register arithmetic can produce 13th place non-zero values. It is important not to confuse the concept of 12 or 13-place precision with the presence or absence of a non-zero 13th place. For example, if e/3 is obtained by the sequence: 1 INV lnx div 3 =, the resulting .9060939428166 is correct to only 11 places, yet there is a non-zero value (6) in the 13th place. Continuing the discussion of his problem with 2 y^x 3 (V1N6p2) Peter Stark notes that the comparison of 8.000000000001 with exactly 8 by display subtraction can produce either zero or a residual -1D-12 depending upon operand ordering. The sequence 2 y^x 3 - 8 = produces zero, while 8 - 2 y^x 3 = produces 1D-12. This is because Reg 60 truncation to 12 places (V1N2p1,2) has no effect on 8, but truncates the 1 in 8.000000000001.

These findings suggest the following programming rules to apply to situations requiring number comparisons: If all numbers concerned can be expected to have 13th place values of zero, display subtractions will produce desired results, otherwise do one of the following: 1) truncate each number to less than 13 places (which Sandy Greenfarb (200) notes can be done by pushing both operands into the pending stack), or 2) perform comparison subtractions by register arithmetic.

A CLEVER D/R SWITCH INTERRUPT PROCESSING APPLICATION

Larry Mayhew (145) has written an SR-52 Timer program that makes effective use of the D/R switch as a means of recording up to 19 time "splits". The nominal constants are for Larry's machine, and may be machine and/or temperature dependent.

SR-52 Program: SR-52 Timer Larry Mayhew (145)

1. Initialize: *fix 4; key nominal constants: 9808.92, press B; 9702.39, press C
2. Press CLR, *CMs; switch D/R switch to degrees, and key start time (HH.mmss); press A, *rset
3. Start "clock" by pressing RUN at step 2 start time
4. Record first (odd numbered) event time(s) by moving D/R Switch to R; record second (Even numbered) event time(s) by moving D/R Switch to D
5. Repeat step 4 for up to 19 splits, noting exact clock time of last one
6. Press HLT; key last-event time, press E, see first split (HH.mmss)
7. Press RUN, see next split; repeat for all splits
8. For new timing exercise, go to step 3

Program Listing

```
000:  1 SUM 68 90 cos *ifzro 000 RCL 68 *IND STO 69 1 SUM 69
022:  1 SUM 68 *pi sin *ifzro 022 RCL 68 *IND STO 69 1 SUM 69 *rset
044:  *LBL *D' *IND *EXC 68 - *IND RCL 69 = +/- div *rtn
059:  *LBL *E' + *IND RCL 68 = STO 66 *IND RCL 69 *ifzro 129
078:  1 SUM 68 SUM 69 RCL 66 *rtn
089:  *LBL E *D.MS STO 64 0 STO 68 1 STO 69 RCL 65 + RCL 00 div RCL 98 =
115:  *D' RCL 99 *E' *D' RCL 98 *E' GTO 115 RCL 64 - *IND RCL 68 =
138:  div RCL 69 = STO 64 0 STO 69 RCL 69 + 1 = X RCL 64 + *IND RCL 69 =
166:  INV *D.MS *IND STO 69 1 SUM 69 *IND RCL 69 INV *ifzro 150
185:  0 STO 69 *IND RCL 69 HLT 1 SUM 69 GTO 189
202:  *LBL A *D.MS STO 65 HLT *LBL B STO 98 HLT *LBL C STO 99 HLT
```