52-NOTES

Volume 1 Number 6 48/48 November 1976

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

DECAPOWER

In the January 1976 issue of 65-NOTES (V3N1p4) Jim Davidson (HP-65 Users Club member #547) suggested the term "decapower" as a descriptor for the power-of-ten multiplier used in scientific notation displays. I'm going to begin using it in place of "exponent" which is technically incorrect, and the letter D to separate the "mantissa" from the decapower for typewritten numbers, as Jim also suggests. For example, 123^{-45} which is displayed in scientific notation as 1.23 -43 will now be written 1.23D-43. Perhaps, as this notation gets more and more usage, the calculator manufacturers will change their keyboard abbreviations. HP's EEX and TI's EE could be changed to ED (for enter decapower).

FORUM

Programming Help: Ray Mackay (358) 32 Woodhouse Road, Doncaster East, Victoria, 3109, Australia needs an SR-52/PC-100 program "...which can analyse at least 32 sample points on a wave form (by Fourier or Chebyshev) and print out the resultant AMPLITUDES and ADVANCE/LAG OF PHASE equal to ½ of the number of sample points... i.e. with 32 sample points 16 harmonics and 16 phase angles. Wave forms must be able to be ODD and EVEN although the programme can be set to run for either case as an alternative." I ask that anyone responding to Ray's request inform me via correspondence copy, as I will want to know how useful such requests for help turn out to be, and may want to share some resulting programs or routines with other members via the Newsletter (see V1N2p4).

Implied Multiplication: Graham Kendall (184) asks for a test to determine whether a particular SR-52 operates with implied multiplication (see V1N5p6). Although I have not seen such a machine, I have reason to believe that there are some, and that the following test should work as an identifier: in RUN mode key: 2 (4 +8) =. For an implied-multiplication machine, the result should be 24; for other machines, the answer would be 12. Anyone having an implied-multiplication SR-52, please let me know.

MACHINE INTERFACE SPECIFICATIONS

Contrary to a purported statement by a TI rep at a recent WESCON meeting (as reported in 65-NOTES V3N7p11), TI is not making public any interface specifications on the PC-100, or the SR-52, for that matter. TI does not encourage the use of any of its machines for other than its designed and published purposes.

TIPS

Manual_Interaction_with_an_Executing_Program: Bob Peirce (121) notes the handiness of manually executed non-numeral keystrokes during the course of program execution at programmed HLTs; (most programs are designed to either display results or receive a manually input number, or both, at a programmed HLT). Bob cites a simple example where, in the business world, commissions are calculated and added to gross on purchases, or deducted from sales, to get net. If commission is in Reg 00 and purchase price or sales price is in Reg 01, the sequence: *LBL E RCL 01 HLT RCL 00 = HLT will handle either case, provided the user keys a + at the first HLT for a purchase, or a - for a sales, followed by RUN to get net. This approach applies to any other functions or sequences that the user might want to key in manually, provided the programmed HLT does not separate steps that must be contiguous. For example, if the sequence: *LBL A RCL HLT is executed, the HLT cancels the effectiveness of the RCL, and a number entered manually is treated as a datum, not the address of a register whose contents is to be retrieved. If you are not sure what effect a programmed HLT might have in connecting programmed execution on both "sides" of a manual entry, try a critical portion of the sequence manually (including the intermediate HLT) and see if each display shows what is intended. Ofcourse, none of this has much practical use if there is enough program memory and a sufficiency of user defined labels to write separate routines for every desired alternative. From the user's viewpoint, the best programs/routines are the easiest/fastest to run, and whose results are the easiest to interpret.

The_Use_of_Inexact_Functions_To_Produce_Integers: Peter Stark (321) laments the frustration encountered when assuming that the SR-52 produces the integer 8 when executing the sequence: 2 y^x 3 =. The resultant 8.000000000001 is certainly close enough to 8 for many applications, but not for others, such as *dsz use in Reg 00, since one trillionth is not small enough to pass for zero, and would cause the *dsz loop to be processed one more time than was presumably intended. In general, if an integer is to be produced by any of the built-in functions: lnx, *log, INV lnx, INV *log, y^x, xrty, *rtx, *1/x or *x! it is best to round it up. The sequence: *fix 0 *D.MS will work for most cases.

Short,_non-Pseudo_Error-State_Producers: Anytime you need to create an error condition in one step, and have not used all ten user-defined labels, a call to an unused one does the trick. For example, the terminating sequence: ...*E' HLT flashes the display, provided *LBL*E' has not been defined anywhere. Then if you need to save even more steps, and can organize things so that your error-producing routine takes up the last few steps, something along the lines of: ...221 RCL, 222 9, 223 8 will work. (Reg 98 needs to contain whatever you want to be flashed). Intentionally allowing program execution to try to proceed past step 223 saves both a HLT and an error producer.

Use_of_Flags_5-9: The discovery that the non-existent flags 5-9 can be put to good use was brought to my attention by T. S. Cox (9) via a copy of a note by S. A. Woods in the 16 Sept 76 issue of ELECTRONICS (p122), and by Stephen Bepko (45). In much the same sense that you can recall zeros from the non-existent Reg 20-59 (V1N1p4) you can test flags 5-9 (which are always off, or unset) either inversely or directly, and cause either a branch or a skip, respectively. Stephen has found a useful application for this discovery in a category of statistics programs, to

cancel entry errors via the same routine that normally processes the entries. Typically, x,y pairs of inputs are summed separately as x, y, x^2, y^2, xy, x^2y^2. If all the required arithmetic is performed in registers, then it can be "undone" by setting a 0 divide 0 error condition (V1N1p2) once, and taking advantage of the viability of the INV prefix (V1N1p2). Stephen suggests the start of a data entry routine along the lines of: *LBL A *ifflg 9 *1' *LBL *2' ... *LBL 1' 0 div 0 = GTO *2'. Data entered for normal processing would be followed by A; data re-entered for deletion would be followed by INV A. The sequence following *LBL*2' would perform all the register arithmetic either normally or during 0 divide 0 error conditions.

An Executable Separation of Register Operator and Operand: In experimenting with unusual STOs and RCLs in RUN mode, James Griggs (13) found the sequences STO SST SST and RCL SST SST somewhat puzzling. What appears to happen is that if the two program steps that would be executed by two successive SSTs contain numeral op-codes, then they are connected to the manually keyed STO or RCL. For example, in LRN mode starting at step 000, key: 1189. Then in RUN mode key: 222 STO 11 CLR *rset RCL SST SST and you will see the 222 retrieved from Reg 11. Now key: STO SST SST CLR RCL 89, and see the 222 that the STO SST SST put into Reg 89. If you try to execute the two SST'd steps automatically, the operator-operand connection is lost.

A New Facet to the CLR Function: During the course of experimenting with pseudos, Ed Haas (187) discovered that execution of CLR does not immediately destroy a "hardened" display (not alterable by keying additional numerals). Apparently, the display register keeps the cleared number until something (other than more CLRSs, CEs, or numerals) hardens a new number. To see how this works, write the sequence *LBL A pseudo 83 CE RCL 60 HLT. Then in RUN mode, key *pi, CLR, A and pi is resurrected. About all you can do between the CLR and the A is to press any number of CLRs, CEs, or numerals without really clearing the input pi. This routine would be handy as part of a program requiring lots of keyed inputs, as a means of retrieving inadvertently cleared or overwritten entries.

ROUTINES

Displaying 11th, 12th and 13th Digits: Graham Kendall (184) has devised a fractured-digits related scheme (see V1N2p5) to display the last 3 hidden digits (following the 4th through 10th): *LBL E STO 99 0 + STO 60 RCL 99 HLT. With the 13-digit number in the display, press E, then =, and see the last ten digits (without any decimal point) in the display. E. L. Parsons (65) accomplishes the same result in fewer steps, but requires creation of a pseudo, unless the sequence is manually keyed: *LBL E + STO 60 = pseudo 31. Again, with the 13-digit number in the display, press E, then LRN, and see the same result. By either method, the resulting display cannot be produced entirely under program control, and cannot be printed.

Timed "Crash": Graham Kendall (184) has discovered that when the SR-52 is commanded to branch indirectly to an out-of-range address, it appears to perform an operation on the out-of-range number, the execution of which varies linearly with the size of the number. To see how this works, in RUN mode key: 1 EE 5 STO 00 *IND GTO 00, and note that it takes about 3 seconds for the display to be flashed. Now try 1 EE 6, and it will take about 32 seconds. Graham ran five trials between 1D5 and 3D6 which indicate that if the operation amounts to a one by one decrement of the number, then a rate of about 32 microseconds per decrement applies.

A DIFFERENCE-TABLE PROGRAM USING THE PC-100 PRINTER

There are probably enough uses of difference tables in applied math and engineering to warrant devoting some 52-NOTES space to this topic. The program that follows demonstrates a convenient way to convert unused program memory into added data input capability, and takes advantage of the SR-52's multiple pointer capability. It should be noted that there is no point in generating the type of mathematical table that provides a function evaluation corresponding to an entry argument, since the SR-52 can compute the function for any specific argument (i.e. why print a table of sines when the machine can compute the sine of any input angle?). But most applications of difference tables involve comparisons among all the table elements, to spot critical characteristics or patterns, and this requires that the tables be presented in their entireties to the user. For those unfamiliar with what I am talking about, perhaps the following example of a difference table will help:

4	5		
9	7	2	
16	9	2	0
25			

The first column is a given string of numbers, the second contains first differences, and is formed by taking the difference of successive pairs of the given numbers; the third by the differences between successive pairs of second column numbers, etc. The following program takes up to 32 input numbers and calculates and prints all possible differences:

SR-52 Program: Difference Tables (with PC-100 Printer) Ed

1. Key first column 1 element, press A; see 1 (indicating first element has been processed), and see printed confirmation of the input.
2. Key ith column 1 element, press RUN; see i displayed and printed confirmation of the input. Repeat for i=2,3,... LT 33
3. Get the difference table printed: press E; printed output is grouped by table column. Each group begins with the column number formatted to 8 decimal places, followed by the differences.

Program Listing

```
000: *LBL A *pap *pap *prt STO 88 88 STO 69 1 STO 64
017: HLT + 1 SUM 69 SUM 64 0 = *IND STO 69 *prt RCL 64 GTO 017
040: *LBL E 1 STO 65 RCL 69 - 1 = STO 68
055: 88 STO 66 89 STO 67 *pap RCL 65 *fix 8 *prt INV *fix RCL 68 STO 64
080: *IND RCL 67 - *IND RCL 66 = *prt *IND STO 66 1 SUM 66 SUM 67
102: INV SUM 64 RCL 64 - 87 = INV *ifzro 080 1 INV SUM 68 SUM 65 RCL 68
129: - 87 = INV *ifzro 055 *pap *pap *pap *pap 0 HLT
```

The "REVISED" SR-52 OWNER's MANUAL

Acting on TI's statement to me concerning the new Owner's Manual (V1N4p1), I ordered one. As I soon discovered, there has been no significant change or addition between the 1220447-1 and 1220447-2C editions. Not even all the errors have been corrected. Apparently, TI meant to convey to me that beginning last August, those buying new SR-52s would get the Programming Workbook (V1N5p2) as a supplement to the Owner's Manual. I regret having disseminated incorrect information, and hope that those of you who have ordered the "new" Owner's Manual are able to exchange it for the Programming Workbook, or get a refund, whichever is desired.

LOCAL CLUBS

The names of and contacts for local calculator clubs that might be of interest to SR-52 or SR-56 users will be noted in this space. The first to come to my attention is CHIP, in the Chicago area. Contact Craig Pearce (18) for information.

REGISTER EXCHANGE FOR DATA RECORDING AND RETRIEVAL

Data storage and retrieval via magnetic cards is a powerful SR-52 capability, especially if optimally programmed. The concept centers on the exchange of progrm and data registers. In most practical applications, data are first produced and stored in data registers, then transferred to program registers whose contents can be stored on mag cards. This process is reversed for data retrieval. Although at first glance it might appear that mechanization would be straight forward, there are trade-offs that should be considered. First, let's take a look at register distribution. There are 60 addressable registers that will hold data, the contents of 28 of which are also capable of being recorded on mag cards. Thus it would appear that a maximum of 28 separate data could be stored and retrieved. But if all 28 program-stored registers were used for data, there would be no room for a transfer program, and only 2 data registers (presumably Reg 60 and 61) would be available to the data-producing program as working/pending arithmetic registers. While we might be able to get by with only Reg 60 and 61 for data production, the requirement to manually transfer data to and from program registers would defeat our primary purpose: to store and retrieve data automatically and accurately. So assuming that we need a transfer program, let's see what it should do, and how to go about optimizing it.

If the transfer program size is to be minimized, it will need a loop within which pointers can be readily "moved". The most efficient way to move the pointers is to increment or decrement them each time the loop is executed. Now, this means that all the program and data registers concerned in the transfer must be consecutively addressable in their respective categories. The program registers present no problem, but it would appear that the longest addressable sequence of data registers is from 98, 99, 100, ... 119 (see V1N1p5 and V1N3p1). These 22 data registers would require a corresponding number of program registers, which would leave 6 (the equivalent of 48 program steps) available for the transfer program. Now let's see if this is sufficient for all that we would like done. In order of priority, here is a likely set of requirements: 1) Transfer data from 22 program registers to 22 data registers, 2) transfer data from 22 data registers to 22 program registers, 3) have the option of transfering only as many data as are needed, and 4) have the program tell us which registers are unused (when we transfer less than 22 data). As minimum, we need 1) and 2), or there is not much point in proceeding. On page 89 of its Programming Workbook (see V1N5p2), TI shows how 1) and 2) can be mechanized, but its program takes 63 steps (15 steps over the 48 available). The resulting partial over-write is acceptable provided the user keeps the transfer routine safely recorded on a "master" card which must be entered each time a new set of data are to be recorded (on a separate card). It would be handier (and require fewer card-reads) if both transfers could be accomplished by the same routine. A program that Ron Zussman (88) wrote back in January 1976 does this, and with a few modifications, does 3) and half of 4):

SR-52 Program: Register Exchange — Zussman/Ed

Program Listing

```
000: *LBL A +/- + 98 STO 69 = STO 68
013: *IND RCL 69 *IND *EXC 68 *IND STO 69 1 SUM 68 SUM 69 97 -
035: RCL 68 = *ifpos 013 RCL 69 HLT
```

SR-52 Program: Register Exchange (con)

User Instructions

To Prepare Mag Card:

1. Run a program that stores data to be recorded in Reg 98-119
2. Enter Register Exchange Program (46 steps starting at step 000), either manually or by card.
3. Key the number of data registers to be exchanged, press A. Address of next available data register is displayed (120 indicates all are used). Data have been transferred.
4. Record (or re-record) card (both sides); note on card how many data have been recorded.

To Use Recorded Data

1. Read card (both sides)
2. Key number of data registers to be exchanged, press A; see displayed address of next available data register.
3. Read (or manually write) program that is to use the data, and run it.

SR-56 PROGRAM EXCHANGE

David W. Johnston (5) is offering to serve as a focal point for the exchange of SR-56 programs. Dave proposes to run this service at minimum cost to users: SASE for a catalog of programs, and SASE plus 5¢ per page (or equivalence in stamps) for requested programs. Program contributors should send Xerox-reproducible copies to both Dave and me. There will be no attempt to referee any programs in any manner, and the only reward to contributors will be the potential satisfaction of sharing their creations.

Dave has already written a number of programs to start things off, and if the other SR-56 members will share their programs, our SR-56 Program Exchange activity has a good chance of succeeding, especially since TI is not providing such a service. I hope that as the Exchange progresses, new programming techniques will evolve that will provide good SR-56 material for the Newsletter.

TABLE LOOKUP OPTIMIZATION ADDENDUM

Graham Kendall (184) has kindly noted an omission in routine C (V1N5p5). The sequence: ... INV SUM 68 X 16 = ... should read: ...INV SUM 68 RCL 68 X 16 = ...

MORE ON DISPLAYING 11th 12th and 13th DIGITS

Putting to use Jared Weinberber's end-of-program LRN discovery (V1n4p6), if Ed Parson's routine E (V1n6p3) is placed in Reg 97, it can be called either by pressing E, or by another program, and the desired last ten digits will be automatically displayed. However, if the call is by another program, there is no execution of a *rtn, and the return-pointer needs to be reset.

MORE ON SHOOTING STARS (V1N4p3)

Stephen Bepko (45) has a 13-shot sequence that shoots out all the stars.