52-NOTES

Volume 1 Number 3 48/48 August 1976

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

PC-100 PRINTER TECHNIQUES

The field is wide open to clever use of known as well as unknown features combining the SR-52 or SR-56 with the PC-100 printer. For instance, small routines could be written specifically to be run in the trace mode in order to produce alpha as well as numeric cues, or to be listed to show 11th, 12th, and 13th mantissa places. Send in your clever ideas that you wish to share.

Over the years, programmers have taken advantage of the way large computers and their line-printers have been designed to get pictures and graphs generated and printed under program control. The primary limitation is that a resolution cell cannot be made smaller than the space occupied by a character. However, line-printer pages are sufficiently large to cover more resolution cells than would generally be needed. While the PC-100 printer can be used this way under SR-52/56 control, the short line-length and limited number of characters present a considerable challenge to the programmer. Look for the program: Printer Graphics (found elsewhere in this issue) which is based on a routine developed by Tony Barlow and Carlisle Phillips, and which shows how the user can get the printer to plot y as a function of x, given x_0, delta x, and the number of points desired. This program has been left in relocatable form, as most users will want to re-format output to suit themselves.

POINTER RULES

To a programmer, a pointer is a register (or other computer device that "holds" a number) in which numbers are stored that are intended to be the addresses of other registers or program steps. Logically, a pointer should only contain positive integers in an "addressable" range. However, the SR-52 can, in some cases, handle a larger set of reals which are "recognized" as proper addresses.

For register pointers, the SR-52 acts only on the tens and units places of a pointer number, and treats any negative number as zero. Thus a large number can be put into a pointer, and used to point to as many as 13 different registers. For example, in run mode, key: 9.876190899, EE 12, STO 01, 180, SUM 01. Reg 01 now points to Reg 00 (put something in Reg 00 and *IND RCL 01 to verify this). Now divide Reg 01 by 10 and you will find that it points to Reg 18; another division by 10 and it points to 91, then to 99, 89, 08, 90, 19, 61, 76, 87, 98, and 9. Further divisions by 10 will make Reg 01 point to Reg 00 each time.

For program step (address) pointers, the entire integer portion of a positive real is acted upon. If this exceeds 223, an error condition is created. Negative numbers are treated as zero.

The SR-52 Users Club is a non-profit loosely organized group of SR-52 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, and a contribution of $6.00 brings the sender six issues of 52-NOTES.

SR-52 Program: Printer Graphics — Barlow, Phillips, Ed

User Instructions

Step	Procedure	Units	Press	Display
1	Enter Program	card	2nd read twice	
2			GTO E	
3	In LRN mode, key f(x) followed by *rtn (assume x in the display)			
4	In run and trace modes, key a nominal x value, then E. See printed the verified f(x) and its value. Get out of trace mode.			
5.	Key x_o			A
6.	Key delta x			RUN
7.	Key number of desired points			RUN
	See max f(x), min f(x) and the plot printed in that order.			

Note: Reg 00, 01, 02, 03, 17, 18, 19, 98, and 99 are used by the plotting program, and should not be used by the function at Label E.

Program Listing

```
000  *LBL A      002  STO 19      005  STO 18      008  HLT
009  STO 17 HLT  013  STO 01      016  - 1 =       019  STO 00
022  RCL 18      025  E           026  STO 02      029  STO 03
032  RCL 17      035  SUM 18      038  *LBL *1'    040  RCL 18
043  E  X        045  (STO -      048  RCL 02)     052  *ifpos *2'
054  1 =         056  STO 02      059  *LBL *3'    061  RCL 17
064  SUM 18 CLR  068  *dsz *1'    070  RCL 03      073  *fix 2 *prt
076  - RCL 02    080  *prt  =     082  *1/x  X     084  10 =
087  STO 98      090  +/-  X      092  RCL 02 =    096  STO 99
099  RCL 01      102  STO 00      105  *LBL *4'    107  RCL 19
110  E  X        112  RCL 98      115  + RCL 99    119  = *fix 0
122  EE INV EE   125  INV *log    127  div 9 =     130  EE INV EE
133  X 8 + 9     137  *1/x =      139  INV *fix    141  *prt
142  RCL 17      145  SUM 19      148  *dsz *4'    150  CLR *rtn
152  *LBL *2'    154  1 = X       157  (STO -      160  RCL 03)
164  INV *ifpos  166  *3' 1 =     169  STO 03      172  GTO *3'
174  *LBL E
```

- - - - - - - - - - - -

LETTER GAME CONVENTIONS

While straight number games pose no symbol convention problems, those involving letters do. I suggest that SR-52 letter games (Hangman, word-Bagels, etc) follow the "Rausch Overlay" convention: Each digit key represents 3 letters, with the user defined function keys: B, C, and D acting as left, center, right "shift" keys that specify which of 3 letters is intended. Thus, the 7 key represents A, B, and C; 8=DEF, 9=GHI, 4=JKL, 5=MNO, 6=PQR, 1=STU, 2=VWX, and 3=YZblank. For example, the keyed sequence: 7,B produces a number in the machine that represents the letter A; 5,C produces N; 1,D produces U, etc. If routines B, C, and D are written consecutively as: *LBL D, +9, *LBL C, +9 =, *LBL B ... then the alphabet translation becomes: A=7, B=16, C=25, D=8, E=17, F=26, G=9, H=18, I=27, J=4, K=13, L=22, M=5, N=14, O=23, P=6, Q=15, R=24, S=1, T=10, U=19, V=2, W=11, X=20, Y=3, Z=12, blank=21. If, for a particular application, letters are to be input in succession, then function B written as: *LBL B, *EXC 05, *EXC 04, *EXC 03, *EXC 02, STO 01, HLT would put the first "letter" in Reg 01, the second in Reg 02... the fifth in Reg 05. If the HLT is replaced by *rset, and the first two program steps are: 000 CLR, 001 HLT, followed by whatever processing is to be done, then a RUN keyed after the last letter is input will initiate main program execution.

THE NIM GAMES

The word NIM (or Nimb) refers to a variety of 2-player contests in which various rules are applied to a playing field consisting of a pile (or piles) of chips. The simplest NIM game begins with one pile of specified size. Players alternate, removing up to a specified maximum number of chips at each turn until the loser is forced to pick up the last chip. In a more sophisticated game, there are 3 or more piles of specified starting sizes (not necessarily the same). Players alternate, removing as many as desired from any one pile until the winner picks up the last chip. I invite members to submit programs for a K-pile NIM game. I will publish the best in a future issue (judged on I/O ease, low execution time, and the size of K). In the meantime, here is a lesser-known NIM type of game which I came across in an exercise in Volume I of Donald Knuth's "The Art of Computer Programming" which he credits as an unnamed game to R E Gaskell and M J Whinihan. I have named my SR-52 mechanization of this game: "Dynamic NIM", since the maximum number of chips a player may remove changes as the game progresses.

SR-52 Program: Dynamic NIM*

User Instructions

STEP	PROCEDURE	INPUTS	PRESS	OUTPUTS
1	Enter Program	card	(2nd read)twice	
2	Key Number of chips	2 LT INT LT 105	A	Pile.Max**
3	Key Player's Move	0 LT INT LE Max**	RUN	Remaining Pile.Nextmax
	repeat step 3 until there is a winner			

* Object of game: take last chip (or all remaining after first move). On first move, player may remove any number, but not all. Machine follows by removing no more than twice what player removed. Play continues with both player and machine restricted to no more than twice the previous move. Intermediate display presents the remaining pile after both player's and machine's moves, as the integer part. Fractional part shows the max** number of chips player may remove on his next move. Flashed max indicates illegal (too large or too small) player's move, in which case the player may try again after pressing CE. When the machine wins, it displays an upsidedown happy message. When player wins, machine flashes its concession of defeat. Non-integer moves (or original pile) result in non-terminating program execution.

Program Listing

000	*LBL A	002	STO 01	005	- 1 =	008	STO 04
011	GTO 190	015	HLT	016	STO 02	019	INV *ifpos
021	161	024	*ifzro 161	028	- RCL 04	032	=
033	+/-	034	INV *ifpos	036	161 RCL 02	042	INV SUM 01
046	RCL 01	049	*ifzro 169	053	RCL 02	056	X 2 -
059	RCL 01 =	063	*ifpos 213	067	RCL 01	070	STO 08
073	0 STO 05	077	1 STO 06	081	RCL 05 +	085	RCL 06 =
089	STO 07	092	*EXC 06	095	STO 05	098	RCL 07
101	- RCL 08	105	=	106	INV *ifpos	108	081
111	*ifzro 126	115	RCL 05	118	INV SUM 08	122	GTO 073
126	RCL 02	129	X 2 -	132	RCL 08 =	136	INV *ifpos
138	181	141	RCL 08	144	X 2 =	147	STO 04
150	RCL 08	153	INV SUM 01	157	GTO 190	161	pseudo 84
162	RCL 04	165	*fix 0	167	GTO 015	171	pseudo 84
172	3507.1	178	*fix 1 HLT	181	2 STO 04	185	1 INV SUM 01
190	RCL 01	193	*ifzro 213	197	+ RCL 04	201	div 1 EE 5 =
206	INV EE	208	*fix 5	210	GTO 015	214	5178.4
220	*fix 2	222	HLT				

DIX FULTON'S 4 X 4 DETERMINANT PROGRAM

Dix's program does indeed appear to do all that was claimed, and is a good example of efficient programming. It also gets high marks for I/O ease, and may leave enough memory space for the possible addition of a Cramer's Rule solution to a system of four simultaneous equations. Dix's condensed listing format is both efficient and readable as applied to this type of program that consists mostly of a long algebraic string. The user must remember where the implied * symbols belong, to designate the 2nd shift key.

SR-52 Program: 4 X 4 Determinant Dix Fulton May 1976

User Instructions

1. Initialize with "E". (Required only for first case)
2. Enter matrix elements in order, each followed by "RUN".
3. Answer appears 8 seconds after last entry.

Program Listing

```
000: LBL A X RCL16 - RCL15 X rtn
012: LBL B X RCL14 - RCL13 X rtn
024: LBL E + 16 STO00 0 =
034: HLT IND STO00 dsz034
043: (RCL11 A RCL12) X (RCL01 X RCL06 - RCL02 X RCL05) +
071: (RCL07 A RCL08) X (RCL02 X RCL09 - RCL01 X RCL10) +
099: (RCL03 A RCL04) X (RCL05 X RCL10 - RCL06 X RCL09) +
127: (RCL09 B RCL10) X (RCL03 X RCL08 - RCL04 X RCL07) +
155: (RCL05 B RCL06) X (RCL04 X RCL11 - RCL03 X RCL12) +
183: (RCL01 B RCL02) X (RCL07 X RCL12 - RCL08 X RCL11  GTO E
```

AUTOMATIC CARD READ

Some members are having difficulty getting programmed *reads to work. The following rules appear to apply, and may be helpful: 1) At any given time (while turned on) the machine is in either of two read states: 1st: the next read will transfer data to steps 000-111, and 2nd: the next read will transfer data to steps 112-223. 2) When executing a *read instruction, the SR-52 transfers the side being read of a card (either A or B) to either locations 000-111 or 112-223, depending upon the current read state of the machine. 3) At turn-on, following a manual or programmed CLR, or following an even number of reads, the machine is in a 1st read state; following an odd number of reads (with no intervening CLRs) the machine is in a 2nd read state (not to be confused with *read), 4) Following a programmed read, execution resumes at the step containing (or that contained) the *read instruction just executed. Some of the implications of these rules are: 1) A *read located at step i in the range 000-111 that is executed when the machine is in a 1st read state will transfer 112 program steps from the card to locations 000-111, and the machine will resume program execution at step i. 2) A *read located as in 1) above, but with the machine in a 2nd read state will transfer 112 program steps from the card to locations 112-223, and the machine will re-execuate the *read (still at step i). 3) A *read located at step j within the range 112-223, that is executed when the machine is in a 1st read state will transfer 112 program steps from the card to locations 000-111, and the machine will re-execute the *read at step j. 4) A *read located as in 3) above, but with the machine in a 2nd read state, will transfer 112 program steps from the card to locations 112-223, and the machine will resume program execution at step j.

Has anyone been successful at getting INV *read to work under program control? (Every time I've tried it, the drive motor runs away!)

ADVANCED PROGRAMMING TECHNIQUES (Part I: ICS)

For the most part, routines presented in this space will not, of themselves, have much direct practical application. They will, however, demonstrate some of the advanced techniques of software mechanization used on large machines. And some will help to optimize practical SR-52 programs.

An Interpretive Computer Simulator (ICS) is a computer program which when run on machine A executes code written for machine B, and to the user makes machine A appear to be machine B, except that execution takes longer. Although the program: "HP-65 ICS" that follows is a very simple ICS, it does introduce some advanced concepts. An operational ICS, besides being able to recognize all possible instructions, needs to scan several at a time to determine context. This SR-52 program can only process three consecutive HP-65 instructions (coded as two or four digit positive integers), such that the first two are recalls from any of the registers 1 through 8, and the third is an operator. But it does demonstrate vectored translation processing, assembling, and loading. One way to process an input instruction code is to run it by a table of values until it is matched. The position in the table of the matching element then determines what processing is to be done. However, if the table is large, considerable search time is required. Vectored processing is direct, and therefore faster. For this exercise, use is made of the fact that all but one of the allowable operator op codes happen to be in an addressable range, i.e. they fall between 0 and 223. It is also convenient that each arithmetic operator is separated from the next by 10. The vectoring is mechanized by an indirect branch through Reg 12. For example, a "71" in HP-65ese represents the X arithmetic operator, and program step 071 begins a sequence to "convert" the 71 to 65 (the equivalent SR-52ese). Note that a 51 causes a branch to step 051 instead of the desired 052. Fortunately, the *2' at step 051 does no harm. The out-of-range 3505 code produces a handy error condition which disables the GTO. In order to simplify processing, I have taken the liberty of merging g y^x as 3505 (instead of the two steps: 35, 05). Three program registers are used in the assembly and loading process. Reg 85 holds a permanent "skeleton" sequence which is transferred to Reg 87 for "fleshing out". Reg 86 contains the unchanging first three steps: *LBL, E, RCL.

User Instructions

1. Key first op code: 340X (X=1,2,...8); press A
2. Key second op code: 340Y (Y= 1,2,...8); press RUN
3. Key third op code: 61=+, 51=-, 71=X, 81=div, 3505=y^x; press RUN.
4. When execution stops, examine SR-52 code following Label E to verify proper translation of HP-65 code.
5. Run function E with appropriate inputs.

Program Listing

```
000  *LBL A      002  INV *fix     004  STO 10     007  HLT
008  STO 11      011  HLT          012  STO 12     015  3400
019  INV SUM 10  023  INV SUM 11   027  RCL 85     030  STO 87
033  *IND GTO 12 037  CE 4.5       041  *LBL *1'   043  EE +/- 10
047  SUM 87      050  GTO *2'      052  CLR 7.5    056  GTO *1'
058  000         061  CLR 8.5      065  GTO *1'    067  0000
071  CLR 6.5     075  GTO *1'      077  0000       081  CLR 5.5
085  GTO *1'     087  *LBL *2'     089  RCL 11     092  EE +/- 5
095  SUM 87      098  10 yx (      102  RCL 10     105  X 10 =
109  *fix 0      111  EE =         113  *PROD 87   116  INV EE CLR
119  HLT         120  000          123  RCL 00     126  = *rtn
128  00000       133  *LBL E       135  RCL
```

LACK OF REG 60-69 ACCESS

A reliable source indicates that TI will recognize and repair without charge an SR-52 under warranty which does not give the user direct access to Reg 60-69. Return your machine with a statement to the effect that all the pending operations functions are not working. Specifically, when the sequence: *pi, STO 60, 0, RCL 60 is keyed, the result is not pi, as it should be. (Be sure the step following STO 60 is "zero", not CLR).

SR-52 MATERIAL IN 65-NOTES

Since a number of members have expressed interest in obtaining copies of 65-NOTES SR-52 material, I have obtained permission from R J Nelson to reprint 22 pages of interest. I will make this material available to SR-52 Users Club members for $3.00 per set. Self addressed labels or large envelopes will expedite turnaround.

PC-100 Hardware Tips

A reliable source suggests removal of the 300 ohm 2-watt resistor on the PC board to reduce the heat buildup. Its only apparent function is to make the VLED power indicator light extinguish a little faster when power is turned off.

Variations in both printer operation and paper quality make print color unpredictable. Users can expect a range from blue to purple to black. Don't use TI 50-50 paper, as it can cause print-head fouling; paper for the silent 700 data terminal may be trimmed to fit the PC-100, and will reportedly produce more consistently black print than the TP-30250 paper specified for the PC-100. Apply the bond-paper head-cleaning method described in the owner's manual twice prior to starting a new full roll of paper.

Note: Members (and other readers) are cautioned that any hardware modifications, product uses, or procedures not specified in the appropriate owner's manuals are performed at the owners/users risk, and may void existing warranties. Neither TI nor the SR-52 Users Club assumes any responsibility for the results of such actions.

Incidently, with the SR-52 in LRN mode, the printer's ADV key will step through program memory, leaving *pap (99) codes in its wake!

DIM DISPLAY

M E Patrick was concerned about the display on his SR-52 becoming dim when .0000001 is entered from the keyboard. TI informed him that this is normal (which is to say that this anomaly is common to all or most SR-52s). A little experimenting shows that it is the 7 places following the decimal point that cause the dimming. Note that the 1 is not dimmed. If a non-numeral key is held down, the display brightens. If some of the zeros are replaced with other numerals, only the leading zeros are dimmed. If an integer part is present, only the decimal point is dimmed.

INTEGER/FRACTION TRUNCATION

Here's a refinement to the *LBL B routine (V1N1p3), contributed by Jared Weinberger: *LBL B, (*fix 0 - ,5), *D.MS, *rtn. The original "STO" can be eliminated, since the *fix 0 plays its "dummy" role. J M Prosser has found a similar double role for the CE key, when an error is to be suppressed under program control at the same time a second display value is needed.

TI's COMPUTER MONITORED REPAIR SERVICE

A reliable source reports that a new automated-tracking repair service for TI machines is in operation at the Lubbock (Texas) facility. Bar-coded stickers are placed on incoming machines, and provide a means to determine in real time, whereabouts and status. Turnarounds are typically four days.