52-NOTES

Volume 2 Number 8 48/39 August 1977

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

ADVANCED PROGRAMMING TECHNIQUES III: SORTING AND SEARCHING

Donald Knuth has devoted a whole volume of his classic "The Art of Computer Programming" texts (Vol 3 Addison-Wesley 1975) to this subject, and I'm not about to try to out-do him! But I do think that many members will profit from being introduced to some of the key concepts, the understanding of which can be useful tools in the generation of many types of non-trivial programs. Although Prof Knuth's text is probably too technical for most non-computer professionals to grasp, fortunately he has written an excellent article on Algorithms (Scientific American April 1977, p63-80) which very nicely discusses many of the important sorting and searching topics at the layman level. Peruse Knuth's article carefully from beginning to end, try mechanizing the algorithms A-F on your PPC, then try running the SR-52 program that follows, and see if/how it implements Algorithms E and F. If you think your program for these algorithms is better (or it applies to a different machine) send it in. Your questions and comments on Knuth's article will form a basis for continuing discussion.

SR-52 Program: Ordered Hashing (Knuth Algorithms E & F) Ed

User Instructions: To Load Hash Table:

1. Initialize: Press CLR
2. Enter word to be loaded: Key letters via Rausch Overlay (V1N3p2), see coded word displayed.
3. Following each completed word, press RUN, see last table-maneuvered word (code).
4. For next word, go to step 1. 32 words max; 5 letters max each.

To Search Hash Table:

1. Initialize: Press CLR
2. Enter word to be searched for: Key letters via Rausch Overlay, see coded word displayed.
3. Initiate search: Press A; if found, see address (88-119); else see input coded word returned.

Program Listing:

```
000: *LBL *C' RCL 68 *rtn *LBL *B' STO 69 *rtn *LBL *D' RCL 69 *rtn
018: *LBL *E' 1 INV SUM 69 *D' - 87 = INV *rtn *LBL D + 9 *LBL C + 9 =
041: *LBL B SUM 69 + *C' X 100 = STO 68 HLT E *LBL *1' *IND *D'
062: *ifzro *2' - *C' = *ifpos *3' *C' *IND *EXC 69 STO 68 *LBL *3'
079: *E' *ifzro *1' 119 *B' GTO *1' *LBL *2' *C' *IND *B' HLT *LBL E
096: *D' div 32 - INV *D.MS INV *D.MS *fix 0 *D.MS INV *fix = X 32 +
115: 87 = *D.MS *B' *rtn *LBL A E *LBL *4' *C' - *IND *D' = ifzro *D'
133: *ifpos *C' *E' *ifzro *4' 119 *B' GTO *4'
```

The SR-52 Users Club is a non-profit Loosely organised group of TI PPC 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 Vandorburgh in Dayton, Ohio. The SR-52 Users Club is neither sponsored nor officially sanctioned by Texas Instruments, Inc. Membership is open to any interested person: $6.00 includes six future issues of 52-NOTES; back issues start June 1976 @ $1.00 each.

CORRECTION TO PROGRAM ON PRECEDING PAGE: Max number of words is 31.

58/59 TIPS

Absolute Addressing: A label search can take as long as 2 seconds (1 second for the 58) for labels near the bottom of program memory, so absolute addressing can often speed execution significantly. It also saves a program step for a branch point to which there is only one branch-to source. However, a single absolute subroutine call to a specified step breaks even with the user-defined-key approach. But since it is handier to call a subroutine from the keyboard with a user defined key than with a sequence of the form: SBR nnn, the added delay may be preferable. However, the ML-02 program shows how to speed things up if you have a few steps to spare by placing called labels near the top of memory, and following some with GTO nnn, where step nnn is the first step of the desired code near the bottom of memory. When converting a program from relative to absolute addressing, keep in mind that each absolute address takes 2 steps: the first holds the MSD and the second the 2 LSDs. Start from the top of program memory, inserting an extra step at each label reference, and deleting each label (2 steps), noting the resulting absolute address. Then go back and overwrite the 2-step address references with the appropriate absolute addresses, making sure they are properly merged.

OP 18 and 19: These two control operations perform the SR-52 iferr function when used in conjunction with flag 7, and are described briefly in the owner's manual (V-29 and V-67). However, the manual does not point out that Op 18 or 19 must be executed each time flag 7 is to be set according to the machine's error state. Thus it appears that each error test should take the form: *Op 19 *ifflg 7 n ... *Lbl n INV *stflg 7 CE ..., and each no-error test the form: *Op 18 *ifflg 7 n CE ...*Lbl n INV *stflg 7 ..., both sequences obviously more cumbersome than the analogous SR-52 sequences: *iferr n... *LBL n CE... and INV *iferr n CE ... *LBL n ... . Incidently, the 58/59 sequence: INV *Op 19 says that if an error condition exists, reset flag 7, and the sequence: INV *Op 18 says that if no error condition exists, reset flag 7. (These appear to be the only 2 control operations affected by an INV prefix).

More on Dsz: The Dsz function can be made to work directly as well as indirectly (V2N7p6) on any addressable register. All that is required is to form a merged 2-digit address by some artificial means: find a key function with the desired op code, or first key a STO, RCL, sum... followed by the 2-digit register address, then replace the STO... with Dsz. For Dsz of a 2-digit register and transfer to an absolute address, first key something like: RCL ab STO cd, where ab is the register to be Dsz'd, and cd the LSDs of the absolute address, then replace the STO with the MSD of the absolute address, and the RCL with Dsz. It appears to have been a design oversight not to have made the Dsz expect to be followed by 2 digits for merging, like the other register commands. Incidently, the flag and fix instructions can also be made to work with 2-digit operands. But there really are only ten flags or display positions and the following occur: *stflg mn sets flag n, *stflg *Ind mn sets the flag identified by the integer LSD of the contents of Reg mn; *fix mn executes as fix n, and *fix *Ind mn fixes the display in accordance with the integer LSD of the contents of Reg mn.

Getting the Correct Memory Partioning: Be sure to mark the prevailing partitioning on a recorded mag card, especially if it is other than the default configuration (479.59) since successful card read and proper execution require correct partitioning, and you can't get the card itself to partition the machine. Incidently, correct partitioning for the V2N6p3 program is 319.79; 239.89 for the V2N7p2 one.

Neutralizing a Label: If a Lbl instruction in a program is preceded by an instruction that expects to be followed by a 2-digit register address or a 3-digit program address, the Lbl (code 76) is treated as Reg 76 or step X76 and the label search mechanism will pass it by. While such a situation is not likely to arise intentionally, hasty editing can produce it. This happened to me once, and at first I thought the machine was at fault, since it wouldn't find a "defined" label whose code I could "verify". In the examples: RCL *Lbl A ..., and *Dsz 6 4 *Lbl B ..., A and B are effectively undefined, since the machine sees these as: RCL 76 A ..., and Dsz 6 476 B respectively.

Out-of-Range Indirect Addressing: Unlike the SR-52, the new machines will not recognize the appropriate LSDs of address pointers that are too large. However, they will operate on the in-range (including partitioning constraints) integer part of a real for indirect addressing. For example, the real: 12.34 can properly point to Reg 12, 123.4 does not point to any register; and 123.45 can point to step 123, while 1234.5 does not point to any step. All negative reals are treated as zero when used as address pointers.

Neutral Error Producer: For n GT 39, Op n produces a non-halting error condition without disturbing the display.

Pause Loop During Error Condition: A pause loop such as: *Lbl A *Pause GTO A executing while an error condition prevails doesn't halt normally with a manual R/S. However, the following seems to work: repeatedly press R/S until the pause changes to a flashed display, then press CE or CLR (a flashed display oscillates faster than the shortest pause loop, and display illumination is dimmer). But if there is any chance of an error condition being set, it is probably best to put a CE before the pause in a pause loop.

FRIENDLY COMPETITION

Hal Brown (HP-65 Users Club member #362) has written an HP-67 5 X 5 matrix program (published in 65-NOTES V4N5p15,16) in an effort to meet the challenge of Barbara Osofsky's SR-52/PC-100 program (V2N5p5). Hal has done well to solve the limited-register problem (the 25 matrix elements take all but one of the addressable registers) and his program appears to work for a few sample problems. It runs fast (gets a 5 X 5 determinant and inverse in less than 3 minutes) and retains the inverse elements in memory, but requires restarts with manually rearranged rows or columns in some cases to get correct inverses, and ofcourse does not print (perhaps someone will write an HP-97 version). I invite members with appropriate matrix algebra expertise and access to an HP-67 to examine Hal's program; send me your findings. Send a SASE to Richard Nelson (2) for a copy, and make the following key entry corrections: step 144 should read RCL I, and step 180 should read RCL (i). In both cases, the key codes are correct. The last part of user instruction 2 should say 5 X 5 (not 4 X 4).

FRIENDLY COMPETITION (con)

So far as I know, no HP-67 user has yet responded to the challenge of Karl Hoppe's 70 Digit Square program (V2N5p5,6). The HP-65 Users Club has proposed 3 HP-25 programs as SR-56 challengers: Gamma Function, Histogram Generator, and 2-way Base Conversion. Write Richard Nelson (2) with a SASE for copies. John Ball (HP-65 member 1345, at Oak Hill Road Harvard, MA 01451) has an HP-25 satellite predictor program that he can't get to fit on an SR-56. Anyone care to try?

GENERAL PURPOSE PLOTTER (59/PC-100A)

While Op 7 makes it easy to position an asterisk in accordance with the magnitude of the number in the display, no other symbols may be substituted, and no other information can be printed on the same line. The program that follows provides for a choice of any of the 64 characters for plot points, and prints each i along the "abscissa" corresponding to the plotted Yi, where Yi=f(Xi). In much the same way as the SR-52 plotter program works (V1N3p2), a first pass is made through a user-defined f(x) to determine YMAX and YMIN. But during this process f(x) evaluations that produce errors are detected, and error messages printed identifying which values of x cause f(x) to produce errors. The plot is scaled to put YMAX at the "top" and YMIN at the "bottom", and ? symbols are plotted where f(x) errors are produced. i-numbers are suppressed when they interfere with plot symbols. Steps 036 and 037 contain the plot-symbol code, which may be changed as desired. I've found that the decimal point (code 40) is especially effective.

Since there is plenty of TI-59 memory to handle this program, I worked more toward optimizing execution speed than toward reducing steps. So 58 users may find it possible to eliminate enough steps to get this program to fit on their machines. The first 20 steps mechanize a vectored processing approach (V1N4p3) to speed up conversion of the display value to a properly positioned symbol code in the print buffer, and the setting of the i-number-suppress flag. Fortunately, the 34 of Op 34 doesn't do any harm when branched to directly (it just wastes a little time taking the square root of 4). Getting the printed i's to increment properly required a bit of counter manipulation (because of the way the numerals are coded); having more than one Dsz register helped. It's too bad the numeral symbols weren't coded with their own decimal values (00-09). The Lbl E R/S at the end of the main program serves only as a quick way for the user to get to the f(x) starting step. His f(x) subroutine is called in the main program via SBR 374. All code above step 374 is absolute, and any editing must take this into account. I looked into the possibility of using ML-07 when f(x) is a polynomial, but there is no way to suppress the Prt at step 076 (the Prt at step 035 can be dodged by calling routine C via SBR 036). I invite refinements and/or better approaches from the membership; simultaneous plotting of 2 or more functions, with different symbols could be a significant enhancement. However, the inclusion of too many goodies could slow execution unacceptably.

On the subject of printer graphics, I should update a statement made in V2N5p2 to the effect that the HP-97 cannot print fractured digits. As revealed in recent issues of 65-NOTES, HP-97 users have discovered a way to get their machines to do some fancy plotting (without burning up the print head). Perhaps this puts the HP-97 somewhere between the SR-52/PC-100 and the TI-59/PC-100A in the Friendly Competition printer graphics arena.

TI-59/PC-100A Program: General Purpose Plotter for y=f(x) Ed

User Instructions:

1. Key E, LRN; write f(x), assuming x is in Reg 55, and end with INVSBR LRN.
2. Key Xo, press A; Xo printed with label.
3. Key delta X, press R/S; delta X printed with label.
4. Key number of desired points (N); N printed with label; N LT 71; see printed YMAX, YMIN, and f(x) plotted with the character whose code is at steps 036 and 037. Along the left margin on the same line as each Yi, i is plotted. Any Xi causing evaluation of f(x) to produce an error condition will generate an error message, and a ? will be plotted for that point. If Xo causes an f(x) error, abort processing with R/S, choose a new Xo value, and go to step 2.

Notes:

1) Xo may be tested by storing it in Reg 55, and pressing E, R/S; Yo is displayed.
2) Registers 5 through 49 and steps 374-479 are available to f(x).

Program Listing:

```
000: *Nop *stflg 1 *Nop *Op 34 *Nop *Nop *Nop *Op 34 *Nop *Nop *Nop
014: *Op 34 *Nop *Nop *Nop *Nop 19 - RCL 58 = div 5 = INV *Int EE
032: 1 INV *Log X 40 = EE INV EE STO 53 1 xEt RCL 04 *x=t 117 RCL 53
053: *Op*Ind 04 *ifflg 1 066 RCL 00 EE 6 = INV EE *Op 01 *Op 05 CLR
071: *Op 20 *Dsz 1 084 2 SUM 00 10 STO 01 *Dsz 2 101 RCL 52 STO 00
092: 100 SUM 52 10 STO 02 INV *stflg 1 RCL 50 SUM 55 *Dsz 3 273 CLR
113: *Adv *Adv *Adv R/S RCL 53 + GTO 055 *Lbl A *CMs STO 54 STO 55
130: 4401 *Op 04 RCL 55 *Op 06 7544 *Op 04 0 R/S STO 50 *Op 06 31
154: *Op 04 0 R/S STO 03 STO 51 *Op 06 *Adv *Adv SBR 374 STO 57
171: STO 59 SBR 374 *Op 19 *ifflg 7 302 xEt RCL 57 xGEt 195 xEt
189: STO 57 xEt GTO 204 RCL 59 INV *xGEt 204 xEt STO 59 RCL 50 SUM 55
208: *Dsz 3 173 19 div (45301344 *Op 04 RCL 57 *Op 06 - 45302431
239: *Op 04 RCL 59 *Op 06 = STO 56 *Adv *Adv RCL 51 STO 03 1 STO 00
257: 7 STO 01 10 STO 02 RCL 54 STO 55 201 STO 52 SBR 374 *Op 19 *ifflg
279: 7 358 - RCL 59 = X RCL 56 = EE INV EE *Op 00 STO 58 4 STO 04
300: GTO*Ind 58 INV *stflg 7 CE 2155445600 *Op 01 1735353235 *Op 02
330: 13370044 *Op 03 64000000 *Op 04 *Op 05 RCL 55 *Prt GTO 204
358: *Op 00 INV *stflg 7 CE 71 *Op 03 GTO 059 *Lbl E R/S
```

- - - - - - - - - -

MORE ON Y^X

Dix Fulton (83) has responded to G E Wilkins' request (V2N7p3) for a general purpose y^x SR-52 routine with: *LBL A (STO 02 X 0 INV *P/R) (*EXC 00 y^x RCL 02) *EXC 00 *P/R *x! *rtn To initialize, store y in Reg 00, x in the display, and switch to radian mode. y^x is returned in Reg 00 following a call to A. If there are no prior pending operations, the two (s can be omitted, and the two)s replaced with =. It appears that if y is negative, odd x cannot be larger than 31. Dix arrived at this clever routine by modifying his V2N2p1 one to meet G E's requirement that a non-integer x with negative y would provide an error condition. Both routines provide for the 0^x error by taking advantage of the fact that SR-52 INV P/R on x=y=0 creates an error. The 56, 58, and 59 do not, and that added to their lack of the x! function suggest a different approach for these machines. Anyone care to try?

EXECUTION OF UNINTENDED CROM CODE (58/59)

On page IV-52 of the owner's manual, TI notes that CROM programs "...do not use = or RST... and end in INVSBR", implying that R/S is not used. Although I haven't found any R/S instructions intentionally written into the ML code, I found a code 91 at step 279 of ML-19, which should behave as R/S if executed by the call: *Pgm 19 SBR 279. However, the machine appears to ignore it. The = (code 95) at step 291 appears to behave normally; the Pause at step 353 appears to be ignored, as does the p21 at step 517. I haven't yet found an artificial RST (code 81) at any of the ML CROM steps... perhaps one of the special purpose applications modules will have one to give us the means to see how a CROM RST would behave. Members finding any other artificial CROM codes not likely to appear intentionally are invited to share their discoveries.

TRIVIA AWARD (52)

Dallas Egbert (384) offers the following discovery as a trivia award candidate: key *read, then simultaneously press the keys: B, INV, sin, STO, EE, 4, and 0. If you've done this correctly, the drive motor should turn on! I wonder if the SR-52 designers have a plausible explanation? Perhaps Dallas' discovery will challenge the even more prodigious prodigy: Bruce Sindlinger, who according to FLYING magazine (p36 June 1977) brought to my attention by Joel Pitcairn (514) apparently did "...most of the design work on the Texas Instruments SR-52" as a teenager, having started college when he was 13. (Dallas, a productive minority member, has devoured about all our local high-school's math dept has to offer, at age 14).

CONVERSION OF SR-52 PROGRAMS TO 59ese

Rusty Wright (581) asks for a few pointers on converting SR-52 programs to TI-59 use, designed for members like him who are unfamiliar with the SR-52. Except for straight-forward number crunchers, it is apt to be risky to attempt to convert instruction by instruction: there are too many functional architecture differences that can produce inefficiencies, or cause real trouble. Although intermediate translation to the algorithm or flowchart level will work in many cases, entirely different approaches will be warranted in others. For example, there is no point in trying to convert Barbara Osofsky's 5 X 5 matrix program (V2N5p5), since 1) ML-02 is always handy, and runs faster, and 2) there are better approaches if you want to do it yourself, given the much larger memory capacity. For many other SR-52 programs, there is no point in copying an approach that uses a lot of subroutine calls to save space, when there is room to put it all in-line, and make it run faster. My advice: make use of routines whose clever features are machine independent, where you can; otherwise start from scratch. Non-SR-52 users can familiarize themselves with SR-52 operation by perusing the owner's manual and/or back issues of 52-NOTES.

58/59 RELATIVE ADDRESSING

Carl Seel (328) asks if the labels referred to in the 2 TI-59 programs (V2N6p3 and V2N7p2) as 1',2',... shouldn't be iffl$, D.MS, ... Well, a rose is a rose..., and I think the 1',2',... nomenclature helps in keeping track of what labels are used, and avoids the confusion of executable functions with labels.