52-NOTES

Volume 1 Number 2 48/48 July 1976

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

- -

THE SR-52 AS AN ADVANCED PROGRAMMING TEACHING TOOL

Members who have taken (or teach) high school or college courses in in computer programming have experienced the frustration of having to share one computer with many users. Long turn-around times are all too common, and tend to discourage programmers from trying refinements past gross executability. Many of the "classical" programming problems can be mechanized on the SR-52, and its dedication to one user helps to motivate him to produce optimum software. Following are some of the advanced programming techniques that come to mind that can be programmed on the SR-52: Binary search; linked lists; manipulation of subscripted variables and arrays; interrupt processing; dynamic code modification; op code translation, link editing, loading, execution; overlays, paging; output graphics (via the PC-100 printer). I expect to discuss some of these in future issues, and invite members to contribute in these topical areas and to suggest additional ones.

JUMPING TO THE WRONG CONCLUSION

As with any detective work that is more than trivial, it is easy to misinterpret machine behavior under uninvestigated conditions. One situation that has frustrated me, and at least one other member is to find that a register which is supposed to contain a non-zero number displays as 0. or -0. The problem is that for a display format current at *fix 0, a non-integer shows up as 0. or -0. In another situation, a member was experimenting with Reg 60 "wipe-out" conditions, and concluded that a certain 8-step sequence would only "work" when it was contained in a single program register called by the sequence. As it turns out, the critical condition was that the first instruction was op code 43, which "transfers" as 42, and it is the 2 in position "B" that causes the "wipe-out" (see part II of Register Behavior below).

REGISTER BEHAVIOR FROM A SOFTWARE VIEWPOINT (PART II)

As many members may have discovered, the reason e-e produces a residual and pi-pi does not is that the 13th place of e is 9, and the 13th place of pi is 0 (roundup from the 14th place). Loss of the 13th place during display (or arithmetic unit) arithmetic is due to the Reg 60 modification of an operand when it is "attached" to an operator. In the Part I example where the "program": B, tan, *rtn, *5', *ifzro, B, tan, *rtn was stored in Reg 70, its modification by Reg 60 following the sequence: RCL 70, - STO, RCL 60, STO 70, produced: AB=34, CD=41, EF=56 GH=78, IJ=90, KL=12, MN=34, OP=56. Note that the original exponent digits: 41 (from D and A) were "dropped down" to positions CD. The 34 at

- -

positions AB combines the attached "-" operator with mantissa/exponent sign information. The remaining digits are unchanged. The important change is the C digit (13th mantissa digit) which got "clobbered" by the MSD of the exponent. Apparently, register arithmetic doesn't work this way, and the 13th place is preserved. Incidently, this discussion concerning Reg 60 applies to Reg 61-69 as well, when numbers attached to operators have been pushed into them by a succession of parentheses. There are additional peculiarities of Reg 60 behavior associated with its role in producing fractured digits and "wipe-out" which are probably due to "invisible" bit patterns. These are the bit combinations in a one-digit 4-bit byte that cannot be set by any of the ten digits. If the ten digits produce the bit patterns: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, then there are six additional bit patterns: 1010, 1011, 1100, 1101, 1110, and 1111 which cannot be user-created, which would look like one of the first ten, but which the machine could and probably does recognize in different ways.

A similar bit-pattern effect probably accounts for the fact that not all program code is "transferable". To see what this means, in LRN mode key the following sequence into locations 000-007: *LBL A, 1, 2, 3, 4, 5, 6. In run mode key: RCL 70, STO 70, and see that locations 000-007 have been changed to: *rtn, 1, A, *1/x, *rtx, *x^2, *stflg, *ifflg. Now re-write 000-007 as: *LBL A, 1, 2, 3, 4, 5, *rtn and you will see that RCL 70 STO 70 doesn't change anything. The first sequence is not transferable; the second is. In this example, it is the number 6 at the 8th step of the first version that causes the transfer problem. The general rule is that the 8th step of any program block of 8 steps cannot be a numeral if proper transfer is to occur. What happens is that the display ignores the leading zero of a coded numeral in the MSD of the mantissa, and shifts the other digits one place to the left. Now put: A, 1, 2, 3, 4, 5, 6, *rtn in locations 000-007, then RCL 70, STO 70. This time the first step A got changed to *E'. This is because position "B" (the LSD of step 000) which carries mantissa and exponent sign information transfers only as 0, 2, 4, or 6. One more complication to keep in mind is that the 0, 2, 4, 6 become 1, 3, 5, 7 if during transfer there exists an error condition produced by a 0 divide 0 or xrty where x=y=0. Apparently, the effect of these two error conditions is to make the LSB of the B position 4-bit byte a one. Thus the 0000, 0010, 0100, and 0110 patterns that normally determine mantissa and exponent sign information become 0001, 0011, 0101, and 0111.

Charles Davis points out that if a number whose mantissa is negative is put into the display (i.e. position B is 2 or 6) the sequence: +, STO 60, = will produce a total "wipe-out" (the effect is to turn the machine off, then on again). However, if the machine is first put into the 0 divide 0 error condition, wipe-out does not occur if the negative number is put into the display by being recalled from a register. The conclusion I reach is that position B must be either 0010 or 0110, which have in common the bit pattern XX10. Incidently, the above +, STO 60, = sequence can be generalized to: A, B 60, =, where A is any of the four arithmetic operators and B is one of: STO, SUM, *PROD, INV SUM, INV *PROD, or *EXC. Further discussion of register behavior will await member response.

WHERE TO BUY MACHINES

While price may be the main consideration, two other criteria should be kept in mind: 1) Delivery time, and 2) service policy. Items that are stocked pose no delivery problem, but it is often difficult to get realistic forecasts on ordered items. From the user's standpoint, the best service policy guarantees him a working machine at all times via a no-cost trade, or loan-while-repair.

HP-65 PROGRAM CONVERSION TO SR-52ese

I'm getting a growing number of inquiries concerning how best to convert HP-65 programs to SR-52 implementation. Unfortunately, there is no magic formula to apply, but I can offer a few guidelines, and welcome additional tips from other members. In general, the more the RPN stack is used (filled and manipulated) the tougher the translation becomes. And, ofcourse, it is one thing just to get a program to run, and quite another to optimize it. The unannounced features of both machines can further complicate things to the extent that even an experienced programmer who is unfamiliar with either machine is apt to run into serious difficulty. Even though I've done a fair amount of HP-65 programming, when I translate other than trivial routines, I first convert the HP-65 program to flow-chart or structured English form, and then work back down through SR-52ese. When there is a lot of stack action, I write down stack contents at each step in order to keep track of changing parameters. It is helpful to keep in mind some basic differences between the RPN stack and pending arithmetic registers: 1) The stack quantities are order-maneuverable, 2) they are not "attached" to operators, 3) as the stack is lowered by 2-number operations, the top-most number is copied downward, and 4) a RCL usually lifts the stack (the top-most number is lost).

An automatic translator that would convert any sequence of HP-65 code to SR-52 code would require considerable programming effort, a large machine to run the translator, and the resulting translation would probably be unacceptably inefficient for practical use. But it might be fun to try!

ROUTINES

Flag_Tester: Bob Dirkman has concocted a space-saving flag-testing routine that is a practical example of dynamic code modification:

```
000  *LBL A      002  RCL 99        005  STO 71     008  *LBL *2'
010  1           011  *ifflg 0 *1'  014  0          015  *LBL *1'
017  HLT         018  1 EE 91       022  +/-        023  INV SUM 71
027  GTO *2'
```

In run mode, initialize with: RCL 71 STO 99. Then key A to see the condition of Flag 0, RUN tests Flag 1, next RUN tests Flag 2, RUN: Flag 3, and RUN: Flag 4. 0 displayed indicates flag is off; 1 that it is set. The effect of steps 018-023 is to increment the digit at step012 (which starts out as 0). If the routine is to be recorded, a few run-time steps may be saved by changing step 002 to RCL 74, and then prior to recording, in run mode key: RCL 71, STO 74 (which needs only to be done once).

D-R Switch_Sensing_(con): Claude Coleman suggests the sequence 90, tan as a test for undesired degree mode, and 0, INV cos, tan for undesired radian mode, both of which create error conditions to signal that the switch is in the undesired position. At first glance, one might wonder why the second sequence wouldn't produce an error condition in radian as well as in degree mode. The reason is that in degree mode: 0, INV cos produces 89.99999999987, while in radian mode it produces pi/2 to 13 places. These two routines save space over ones that test for zero and branch accordingly, but should not be used in programs that process D-R Switch-sensitive data prior to a HLT or *rtn that signals the error.

Popular Computing's July issue devotes a couple of pages to "SR-52 Notes" in which the idea of the D-R Switch being considered as an interrupt processor is discussed. This has applications in both teaching and as a proctical means of monitoring the "progress" of lengthy iterations. As many have discovered, keying HLT during program execution, followed by RUN does not always produce intended execution. The sequence:

*pi, sin, *ifzro, *pi inserted at an interruptable point in a loop, with: *LBL *pi, HLT outside the loop provides for uninterrupted loop execution when in degree mode, and a safe interrupt when in radian mode.

FORUM

Member comments, opinions, suggestions, gripes, etc. not covered elsewhere are aired in this space.

Dix Fulton laments the low quality of TI's software support for the SR-52... a complaint shared by many pocket programmables owners/users against the manufacturers. I suspect that it is just a matter of basic economics: the cost of high quality software would drive prices higher than manufacturers think users/owners would be willing to pay. When it is a labor of love, the long hours devoted to ultimate optimization are spent willingly, but when expensive software expertise has to be paid for, it is not likely to be cost effective, especially for the obsolete-prone pocket-programmables. But users organizations can and should fill the gap between mediocre commercial software, and what the owner/user wants/needs.

Dix notes that three of the four possible fractured digit configurations (see V1N1p3) can be thought of as degrees, minutes and seconds symbols. Charles Davis has devised a way to combine the degree and minute symbols with calculated values, and thus opens up a promising new feature: variations in display formatting. (More on this later) Has anyone been able to create fractured digits completely under program control, i. e. without a manually keyed = (or SST'd =)?

Dix claims to have a "conservative" 4X4 Determinant program that uses only Reg 00-19, executes in 8 seconds, and "never fails to solve". If Dix would like to share it, I'll publish it.

Rather than publish lists of programs members have written, I will put in this space specific requests for help. State the problem you are trying to solve, and the programming aspect(s) giving you trouble: algorithms, keystroke mechanization, I/O handling, etc. Other willing and able members are invited to respond.

Edward Haas has been exploring the behavior of pseudo instructions, and notes some results different from what I have observed. Perhaps there is individual machine dependence, or maybe a state-of-battery-charge dependence involved. Other members are invited to share their experience with pseudos (codes: 21, 26, 31, 61, 66, 71, 76, 62, 63, 64, 72, 73, 74, 82, 83, 84, and 92; defined as code that is unkeyable in LRN mode). I'll summarize results in a future issue. Pseudos are creatable as data: 8.4 STO 70 puts code 84 at step 007.

Cleon Dean notes that SR-52 RF can be picked up by a small AM radio, especially when the display is flashing.

Val Barron has been experimenting with "long" mag cards, and has found that the 10½" card made by HP (part #9162-0045) for its desk-top model 10 machine can be trimmed to become three continuous SR-52 cards. Once he got write-protect black tabs in correct places, Val reports success at executing successive reads under program control. Claude Coleman reports being able to scotch-tape as many as four regular cards in series, and have them all read and executed, He says the trick is to line up (overlap) the end vertical lines (printed on each card) before taping. Anyone trying this should be aware of the risk of read/write unit fouling by the tape and/or overlapping.

INT/FRAC TRUNCATION (con)

The "C" routine cited in V1N1p3 is subject to the condition that at the time it is invoked, the display format must be at least "one" (*fix 1\ Richard Bazan brought this to my attention when he found that the routine always worked the first time, but not always after that. The problem is caused by the *fix 0 in the routine itself.

DISPLAY FORMAT VARIATIONS

In devising a way to get degrees and minutes symbols into the display in desired places relative to calculated values, Charles Davis applied a useful sequence of the form: RCL 01, + StO 60, RCL 02 =. I have expanded its application to produce a wide range of display format options which may be used to enhance a variety of outputs: numerical calculations, game scores, upside-down messages, etc.

It appears that the "+" in this sequence sets some invisible bit patterns in Reg 60 (or somewhere?!) which are not altered by the subsequent STO 60. Reg 01 acts as an "operator" on Reg 02 in the sense that digit by digit, Reg 01 "lets pass" Reg 02 digits or produces fractured digits in their place. If Reg 01 digits are labeled AB, CD, ... OP (see V1N1p5) and Reg 02 digits labeled A'B', C'D', ... O'P', then the following digits are "paired" in the sense that an unprimed digit "operates" on a primed one: CD', DA', EE', FF', GG', HH', II', JJ', KK', and NN'. The contents of Reg 01 are treated in Reg 60 as operator-reformatted, i.e. the positions CD are the exponent (see V1N2p1). Thus the CD quantities affect the D'A' exponent values in Reg 02. The effect of operator digits is as follows:

Digit:	0	1	2	3	4	5	6	7	8	9
Effect:	none	none	"	blank	'	°	-	blank	none	none

If we number display positions from left to right as 0 thru 13, then the resulting digits (or fractured digits) fill the display as follows: positions 0 and 11 get minus signs or not according to the value of B' (which is not operated upon) in the usual way (see V1N1p5). Position 1 gets NN', 2 gets KK', 3 gets LL', 4 gets II', 5 gets JJ', 6 gets GG', 7 gets HH', 8 gets EE', 9 gets FF', 10 gets C' (which is not operated upon), 12 gets CD' and 13 gets DA'. Note that MOP and M'O'P' appear to be ignored. However, neither O nor O' may be 0, as this would make Reg 01 and Reg 02 non-transferable (see V1N2p2), and if M' is zero, for mantissa operands the fracture rules get changed: as operators, the digits 0, 1, 2, and 4 act like 3; 5 acts like 6; while 3, 6, 7, 8, and 9 appear to act normally. The same is true of the exponent operands: A' and D'. If A' is zero, position 13 is affected, and if D' is zero, position 12 is affected (by the change in fracture rules). However, if C, D, A', D' are all zero, then nothing is displayed at positions 12 and 13. The minus sign at position 11 can only be made to appear if either A' or D' is non-zero. The mantissa operands: E', F', G', H', I', J', K', L', and N' appear to introduce the further complication that any non-zero operand "normalizes" the fracture rules for itself and all lower-order mantissa operands. For example, non-zero H' makes E', F', and G' behave "normally", even if they are zero.

AB have no apparent effect on Reg 02, except that if B transfers as 2 or 6, there is the "wipe-out" effect (see V2N2p2), and if B transfers as 4, the intended paired operations do not occur. I have not yet found a practical use for AB being other than 00.

Putting some of all this to use, let's suppose that a program produces four 2-digit positive integer outputs: 12, 34, 56, 78 which are associated in pairs such that it would be convenient to display the output as: 12-34 56-78. To accomplish this, prepare Reg 01 with AB=00, CD=00, EF=30, GH=93, IJ=69, KL=99, MN=93, OP=99. (One way to do this is in LRN mode, beginning at step 000, key: 0, 0, *rtx, ., *9', *pap, ., *pap; then in run mode: RCL 70, STO 01). For Reg 02: AB=84, C'D'=67, E'F'=05, G'H'=40, I'J'=03, K'L'=12, M'N'=99, O'P'=99. (one way to get the (84) at step 000 is in run mode to key 1 EE +/- 8 STO 70). Now, in run mode key: RCL 01, + STO 60, RCL 02 =, and you should see: 12-34 56-78. In this example, "3's" were positioned in Reg 01 to cause desired blanks. The "-" between the 12-34 was created with the 6 at position I in Reg 01, while the "-" between 56-78 is "naturally there in the makeup of Reg 02 (position B' is a 4). In preparing other formats, keep in mind the limitations that display positions 0 and 11 can only be either blank or the minus sign, and that position 10 will be the unmodified C' digit (since it cannot be fractured). For a practical application, it would be necessary to "build" Reg 02 from computed values, under program control. Since it is the contents of Reg 02 that the user sees displayed before he manually keys =, the three high-order mantissa digits may be chosen along with an appropriate *fix to serve as a cue.

56-NOTES

Material is starting to come in from SR-56 owners/users. Since I don't have access to the SR-56 myself, I will publish items sent in, usually without comment. As with the SR-52 material, priority will be given to short routines and discoveries that introduce clever ideas. Keystroke sequences should include step numbers.

Pause Key: D W Johnston notes that the pause function can be used to watch the progress of a well-exercised loop, since a critical (presumably changing) quantity can be briefly displayed at the end of each cycle with the pause instruction.

Zero Divide Zero: D W also notes that similar to the SR-52 case, the sequence: 0, divide, 0, =, CE produces an error condition that causes SUM to execute as INV SUM, (but PROD doesn't execute as INV PROD).

BAGELS ONE TO TEN ANOMALY (see program in V1N1p6)

Larry Mayhew points out that if a ten digit number is being scanned by a guess that has all the digits in wrong places, the score is 1.0 (instead of 0.10). This occurs because ten .1's add up to 1.0, and could be remedied by reformatting the score to read: FF.pp.

NOTATIONS/CONVENTIONS

Notations, abbreviations, acronyms, etc that may not be generally recognized will be explained in English. Let me know what you find puzzling, and I will explain in the next issue of 52-NOTES. A few abbreviations that have cropped up so far are: GT=greater than, LT=less than, GE=greater than or equal to, LE=less than or equal to, *rtx=square root of x, xrty=xth root of y, *pi=3.14..., e=2.718..., div=divide.

MEMBERSHIP

Several members have contributed more than the suggested $6.00 for six issues, and I am grateful. Copying, stamps, phone tolls, etc. do add up. At this writing (10 July 76) there are 171 members, with the number growing every day. I plan to include a membership list in the August issue that goes to members, and remind those who wish to have their names removed to so inform me by the end of July.

MACHINE DIFFERENCES

Fred Gruenberger and Cy Pizette both bought their SR-52s about 3 months ago, and both report lack of access to Reg 60-69. So far, TI's official position is that all SR-52s have been manufactured to the same specifications. If others are experiencing this (or any other unannounced feature) deficiency, identify the problem, and your machine's serial number, and I will try to persuade TI to at least recognize that they have introduced hardware differences.

HP-67

The successor to the HP-65 looks like a close competitor for the SR-52. My first impression is that it is superior as a statistical tool, but that the restrictions on flag, register, and indirect addressing manipulations leave the SR-52 in the lead for programming versatility.