Volume 1 Number 1 48/48 June 1976

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

SR-52 Users Club Scope

Club activity centers on contributions to and perusal of the monthly newsletter 52-NOTES. The club does not have legal status or formal structure, and sole responsibility for its operation is taken by the Editor.

The club is neither sponsored nor officially sanctioned by Texas Instruments (TI), but is recognized by TI as the first independent group of SR-52 owners/users. 52-NOTES covers SR-52 care, use, and performance, as well as material on how best to program it for various applications. Although some complete programs not earlier submitted to TI for its soon-to-be operating Users Library will be included in the newsletter, clever routines will generally be given priority. Other pocket programmables may be considered as newsletter topics, depending on membership interest. As a start scope will include the SR-52 and SR-56. Material exclusively covering Hewlett-Packard pocket programmables will be referred to the HP-65 Users Club.

Newsletter content covers a broad range of technical disciplines and sophistication, and while each contributor is asked to make a reasonable effort to assure that his material is technically correct, neither the Editor nor contributors assume formal responsibility for any consequences resulting from any use of newsletter material. New discoveries will be credited to the first known discoverers, and those knowingly using another's discovery are asked to give due credit. Newsletter material is not copyrighted, and may be copied and distributed without limitation provided excerpted material is not taken out of context and is given due credit.

Routine/Program Listing Format

Your comments are solicited regarding the format used in this first issue for routine and program listings. The idea, of course, is to minimize space and maximize readability and checkability. Step numbers apply to the first instruction in a group. An asterisk signifies 2nd function. Left to right ordering facilitates "one pass" page organization.

Unannounced Features

Most of you know by now that the SR-52 has capabilities beyond those announced by TI. Although TI is aware of some of those, it is likely to support only those it announces. This means that future SR-52 production units won't necessarily have all of the current features. Since the club's purpose is to help members to get more out of their machines, topics will not be limited to announced features and their implementation. However, all contributors are asked to include in their shared routines

---

The SR-52 Users Club is a non-profit loosely organised 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.

and programs a statement that tells which (if any) unannounced features are used.

Following is a list of those unannounced features currently known to the Editor, a brief description of what they are, and the names of those who first brought them to his attention. More detailed explanations are begun elsewhere in this issue, and will continue in future issues.

1. **Registers 60 through 69:** These are used by the machine to hold pending operations, but may be used for data storage with the STO and RCL functions. (Mike Louder)
2. **Registers 70 through 97:** These are used by the machine to hold program instructions, but may also be used for data storage with the STO and RCL functions. (Mike Louder)
3. **Registers 98 and 99:** Extra storage resisters not affected by either the CLR or *CMs functions. (Mike Louder)
4. **Transferable Code:** Machine executable code that is unaltered when transferred between data and program registers. (Mike Louder)
5. **Split INV Functions:** Viability of the INV function as a prefix, even though separated by labels and subroutine calls. (Mike Louder and Sandy Greenfarb)
6. **Multiple INV prefixes:** An odd numbered succession of INV prefixes produces the INV function. An even number cancels it. (Ed)
7. **Pseudos:** Synthetic generation of instruction code not keyable. (Ed)
8. **Register 60 Strange Effects:** Fractured digits and complete "wipe-out" can be produced by certain register arithmetic operations in Reg 60. (Joel Rice and Sandy Greenfarb)
9. **Pseudo 83 and 84 Arithmetic Strange Effects:** Same as for Reg 60 arithmetic. (Ed)
10. **Use of *D.MS Function for Integer/Fraction Truncation:** The sequence *fix, 0, *D.MS rounds up a real into an integer. (Ed)
11. **Use of INV *D.MS to Reduce Reals for Truncation:** The sequence: INV, *D.MS, INV, *D.MS reduces the fractional part of a real to less than one half. (Phil Sturmfels)
12. **Zero Divide Error Conditions:** The sequence: *0, divide, 0, =, CE* generates a unique error condition causing the SUM and *PROD functions to execute as INV SUM and INV *PROD, respectively, and causing a change in transferable code rules. (Ed)
13. **Program Execution of Card Read:** One side of a pre-recorded magnetic card can be read under program control. (Ed)
14. **13-Place Arithmetic:** While limited to 12 places for "display" arithmetic, the machine preserves 13 places in register arithmetic. (Ed)

Games

Many of you know the fun and pleasure that programming and playing computer games can bring. The SR-52's indirect addressing and comparatively large data storage capacity (for a pocket programmable) give it important advantages to be exploited, especially in handling generalized games. However, the lack of built-in integer/fraction truncation functions is often a disadvantage that needs to be overcome. Also relational tests on two values are slow, and destroy the compared values.

My "Bagels One to Ten" program (found elsewhere in this issue) is an example of a generalized Bagels game. I'll publish a "Dynamic NIM" game to play against the machine, which is the toughest NIM I've yet come across. In the meantime, I invite you to send in your best games for possible newsletter publication.

Routines

The more a routine (short program or function) is used, the more its execution and memory efficiency matter. The sharing of routines in this space will help all of us write better programs. I'll kick off with a few that have come to my attention, or that I have evolved, and look forward to seeing many member contributions.

Integer/fraction truncation:

I expect that so far more users have spent more time trying to perfect integer/fraction truncation routines than any other. And for good reason: programs that need these functions are apt to need all the memory space they can get, and are also apt to cycle through the int/frac routines often enough that execution inefficiencies are magnified noticeably. It appears that the routines that execute the fastest also take up the most program memory. Length and complexity of these routines depend upon the range of situations prevalent when they are invoked. The simplest case is that where only positive reals and a non-"scientific" display prevail during its execution. For this case the sequence: ***LBL A**, (STO - .5), *fix 0, EE, INV EE, *rtn* is the fastest to execute, but takes 14 program steps. ***LBL B**, (STO - .5), *fix 0, *D.MS, *rtn* saves two steps, but is a bit slower. ***LBL C**, INV *D.MS, INV *D.MS, *fix 0, *D.MS, *rtn* saves two more steps, but is slower yet to execute. However, Routine C will handle negative and zero reals besides positive ones. If zero must be accounted for as a possible input argument, then Routines A and B would require a zero test, and would lose to C. All three routines would require INV EE to start with if the display format is "scientific" when they are invoked, and need to end with *fix 9 (or INV *fix) if display format must be returned to "initial mantissa". All things considered, Routine C looks like the best, unless speed is at a premium. But don't assume these are the best there are... by all means send in your better ones for publication!

Automatic Fill of Reg 60 - Reg 69 with a Single Number:

Key: ***LBL E**, STO, X, (, INV *if err, E, CE, *rtn* in LRN mode. In run mode: Key n, E; result: there are ten "copies" of n in Reg 60 through Reg 69 plus the "original" in the display.

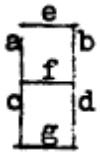
Note: Each Reg 6X n is "attached" to a "X" operator.

Short Fibonacci Sequences:

In LRN mode starting at loc 000 key: *HLT, +, *EXC 69, *rset.* In run mode key: *rset, SST, CLR 1, [:RUN:] *Note:* [:A:] means keep repeating A until "done". The first press of RUN produces F_0=0; each successive RUN produces the next Fibonacci Number. F_{50}= 12586269025. This routine could be made subroutine callable by writing it: ***LBL E**, +, *EXC 69, *rtn*, in which CLR, 1 should precede the call.

Display Fun

Several SR-52 users have already discovered that the display can be made to present "fractured" digits. If digit composition is labeled as follows:



the following four patterns can be generated either by pseudo 83 or 84 arithmetic, or by Reg 60 manipulation: ab, a, abef, f. So far, I've been able to create some symmetrical pattern groupings which can be randomly produced by the following program:

SR-52 Program: Dynamic Display Fun* .

User Instructions

Step	Procedure	Units	Press	Display
1	Enter Program	card	2nd read twice	
2	Key Random Number Seed (r)	0<=r<=10	A	0
3	Get symmetric fractured digit	Display	=	??
4	Start next display		RUN	0
	Repeat steps 3 and 4 as often as desired			

*This program requires access to Reg 60 through 99.

Program Listing

000	***LBL A**	002	STO 99	005	***LBL *1'**	007	*D' B C E
011	E E E C	015	B E C E	019	E C B E	023	E C *C' HLT
027	*D' B C E	031	C E C E	035	C E C *C'	039	HLT *D' B C E
044	E E E C	048	B E C E	052	C E C *C'	056	HLT *D' B C
060	E C E E	064	C E C B	068	E E C *C'	072	HLT GTO *1'
075	***LBL B**	077	RCL 99	080	X *pi =	083	*rt x - D =
087	ST0 99	090	E D	092	ST0 98	095	- 7 =
098	*ifpos B	100	1 -	102	RCL 98	105	=
106	*ifpos B	108	RCL 98	111	*rtn	112	***LBL C**
114	SUM 92	117	*rtn	118	***LBL D**	120	*Ifzro
121	*ifzro	122	(STO - .5)	128	*fix 0	130	*D.MS
131	***LBL *ifzro**	133	*rtn	134	***LBL *C'**	136	RCL 92
139	+	140	SUM 60	143	+ 0 *rtn	146	***LBL *D'**
148	RCL 91	151	STO 92	154	*rtn	155	***LBL E**
157	X 10 =	161	*rtn	162	***LBL *E'**	164	*IND
165	*EXC 98	168	*E' A	170	0 0 0	173	C A A

Register Behavior From A Software Viewpoint (Part I)

Judging from many of your questions already, I sense that register behavior is of considerable concern to many SR-52 users. The discussion that follows is based on input from Mike Louder and Carlisle Phillips, as well as on my own experience.

Even without detailed knowledge of the hardware design, the user can discover new things about register behavior just by observing results under carefully contrived circumstances. Take the manual sequence: *1, INV lnx, -, STO, =.* Where does the resulting -9 -12 come from? Now try **pi, -, STO, =.* Why do you get zero this time, but not for e? (*Note: n-STO= is equivalent to n-n=*). Now try *1, INV lnx, STO 01, INV SUM 01, RCL 01.* This time there is no residual.

Why? In order to find the answers, we need to make use of the display in both the LRN and calculate (run) modes. By performing STO and RCL of data and instruction codes through the potentially addressable 00-99 range, many of you have arrived at the following five classes of registers: General Data: 0-19, Read-Only Zeros: 20-59, Pending Arithmetic: 60-69, Program Memory: 70-97, and non-clearing Data: 98 and 99. You soon discover that Reg 70-97 in LRN mode display the most information, and it becomes evident that each register (in any category) may be thought of as a 64-bit binary word, divided into 16 4-bit bytes.

As a convention that should be easy for all to follow, write the sequence starting at location 000 in LRN mode: B, tan, *rtn, *5', *ifzro, B, tan, *rtn. The resulting code should look like: 000 12, 001 34, 002 56, 003 78, 004 90, 005 12, 006 34, 007 56. Label the 16

instruction code digits as follows: 12=AB, 34=CD, 56=EF, 78=GH 90=IJ, 12 (at loc 005)=KL, 34=MN, and the last 56=OP. Now switch to run mode and RCL 70.
-5.634129079 41 is displayed, which bears some resemblance to the 8 op codes, but not much! Now STO 71, and examine steps 008-015 in LRN mode. They should represent the same "program" as in steps 000-007. Apparently all 16 digits got transferred, even though only 12 showed in the display in run mode.
Here's the key: by the above convention, OP are the highest order displayed mantissa digits, followed by MN, KL, IJ, and GH. EF and C are the 11, 12 and 13th digits, respectively. D is the higher order exponent digit, A the lower order one, and B contains information determining mantissa and exponent signs as follows: 0=++, 2=+-, 4=-+, and 6=--. Now RCL 70 again, and see if the displayed number makes sense, keeping in mind that the 11, 12, and 13th places round up in the display to the 10th, Now key: -, STO, RCL 60, STO 70, and look at steps 000-007. Can you see what Reg 60 did to the displayed number in "attaching" it to the "-" operator? Part II in a later issue will continue register behavior discussion.

Programming Tips

Here are a few programming tips that may be new to many of you. If you have others you'd like to share, send them in for possible publication. This first group was brought to my attention by Carlisle Phillips:

22 Sequential Data Registers: Reg 98, 99, 00, 01, ...19 may be addressed indirectly consecutively as 98, 99, 100, 101, ...119, extending the usual 01-19 sequence by two.

Fast Execution by Absolute Addressing: Conditional or unconditional branches performed many times will execute in noticeably less time if they are in absolute form. The time saved is greatest for branch-to points near the end of a program. Best approach is to debug a program using relative (label) addressing. Then when it runs satisfactorily, remove the labels and replace their symbolic references by 3-digit absolute addresses, keeping in mind that a single symbolic reference to a label takes the same number of program steps as an absolute branch, i.e. 3. (see the program "Bagels One to Ten" found elsewhere in this issue). Incidently, those of you who plan to send in games for publication should optimize execution speed by maximizing absolute addressing.

LBL LBL Tricks: The ability of the LBL function to serve simultaneously as both a label identifier and as a label itself may be exploited to space-saving advantage. The sequence *LBL LBL --- GTO LBL, A* --- defines both the loop LBL --- GTO LBL and a label called "A". The sequence: *LBL LBL func --- *rtn* defines two different subroutines: 1) called by SBR LBL that does func followed by ---, and 2) called by SBR func that does only ---.

Run-Time D-R Switch Sensing: For any program that is sensitive to the D-R switch position, and which has sufficient unused program memory, the following routines will alert the user to improper switch position: For desired degree mode: **pi, sin, *ifzro, *pi ---- ***LBL *pi** (84), 0, HLT* will flash zero if switch is in radian mode. For desired radian mode: **pi, sin, INV *ifzro, *pi ---- ***LBL *pi**, (84), 0, HLT* will flash zero if switch is in degree mode. Note: use of pseudo 84 creates an error condition in one step.

SR-52 Program: Bagels One To Ten* R.C. Vanderburgh

User Instructions

Step	Procedure	Units	Press	Display
1	Enter Program	card	2nd read twice	
2	Key number size (n*)	0 < n < 11	D	
3	Key Random Number seed	Positive Real	RUN	
4	Generate Mystery Number		E	0.
5	Guess Number	Integer*	A	Score*

For new guess, repeat step 5.
For new mystery number, go to step 4.
For new number size, do step 2, then skip to step 4.

*Object of this game is to guess an n-digit positive integer by scanning with n-digit positive integers. Player chooses the size of n at step 2. Mystery integers contain no repeated digits, and thus the step 4 execution time will vary as a random number generator does or does not produce repeating digits, as well as upon integer size. Integer part of score gives the number of correct digits in correct places (fermi). Fractional part of score gives the number of correct digits in wrong places (picos).

Program Listing

```
000  *LBL *D'     002  RCL 14      005  X *pi =     008 *rt x - *E' =
012  STO 14       015 X 10 =       019  *E'         020 STO 12 *rtn
024  *LBL *E'     026 *ifzro       027  039         030 (STO - .5)
036  *fix 0       038 *D.MS *rtn   040  *LBL E      042 2 STO 11
046  *D' RCL 12   050 STO 01       053  1 STO 13    057 *D' RCL 12
061  - *IND RCL13 066 = *ifzro     068  053         071 1 SUM 13
075  RCL 13 -     079 RCL 11 =     083  INV *ifzro  085 058
088  RCL 12       091 *IND STO 11  095  1 SUM 11    099 RCL 15 -
103  RCL 11 =     107 *ifpos       108  053         111 CLR *rtn
113  *LBL D       115 STO 15 HLT   119  STO 14      122 CLR *rtn
124  *LBL A       126 STO 16 CLR   130  STO 17      133 RCL 15
136  STO 00       139 RCL 15       142  STO 11      145 RCL 16
148  div 10 =     152 - *E'        154  STO 16 =    158 X 10 =
162  STO 12       165 RCL 12 -     169  *IND RCL 11 173 = *ifzro
175  201          178 1 INV        180  SUM 11      183 RCL 11
186  INV *ifzro   188 165          191  *dsz        192 139
195  RCL 17       198 *fix 1 *rtn  201  RCL 00 -    205 RCL 11 =
209  *ifzro       210 214          213  .1          215 SUM 17
218  GTO 191
```

Miscellaneous

TI Notes: Extra copies of the SR-52 Owner's Manual and Basic Library are available from TI at $4.95 and $3.50, respectively. (For Carl French: TI is looking into your unfilled order problem). Four applications program packages are currently available at $29.95 each: Math, Statistics, Finance, and Electrical Engineering. Sets of 40 blank mag cards with carrying case and black tabs are available at $15.95 each. Call TI Customer Relations at toll free 800-527-4980 or write to P O Box 22283 Dallas, TX 75222. The idea of a TI sponsored SR-52 Users Library is currently in limbo. The PC-100 printer is being shipped to distributors in small quantities.

Membership List: Members who do not wish to have their names, addresses, etc. to appear on a distributed membership list should so indicate to the Editor by the end of July. Letters to the Editor that require individual reply should include a SASE.

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 13^{th} 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, *$\sqrt{x}$, *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-charter 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 step 012 (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 practical 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 interruptible 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 Barren 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', EF', 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
Effects:	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 N' 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 l2 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 V1N2p2), 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 56-73. To accomplish this, prepare Reg 01 with AB=00, CD=00, EF=3O, 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, *√x, ., *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, *$\sqrt{x}$= square root of x, $\sqrt[x]{y}$=x^{th} 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.

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.

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_0		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  SUN 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=GH1, 4=JKL, 5=NNO, 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; 1,0 produces U, etc. If routines B, C, and 0 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, L=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 N 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 10^5	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 ST0 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 STO 00 0 =
034:  HLT IND STO 00 dsz 034
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 stop 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-execute 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 y^x (      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.

Volume 1 Number 4 48/48 September 1976

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

TI Announces its Professional Program Exchange (PPX) for the SR-52

$15.00 sent annually to TI entitles the sender to PPX membership, a catalog, and three programs. Additional programs cost $3.00 each. PPX members will receive one free program and several (probably four) mag cards for each program they submit that is accepted by TI. The first catalog will list several hundred (maybe a thousand) programs.

New SR-52 Owners Manual

A new owners manual is now available from TI for $4.95 a copy, which recognizes and discusses previously unannounced features. See the TI advertisement in the September 1976 issue of **Scientific American** magazine for details about previously unannounced features, and the PPX service.

Editorial

Now that TI has announced its PPX-52 service, it will be my policy not to publish programs that have been (or are planned to be) submitted to TI. I will continue to give priority to clever routines and programs that demonstrate new techniques over those that are mechanized in well known ways. Unpublished programs will be returned to members who so request, and who send SASEs.

More On LBL LBL

Jared Weinberger (221) adds another LBL LBL trick with the sequence: **ifflg 2, *LBL *LBL, *√x...,* which does the same thing as: *INV, *ifflg 2, *1, *√x, *LBL *1'*,... for a savings of two steps. This use of LBL LBL can be applied to any of the conditional branch statements where a single instruction is to be executed for one condition, and not for the other.

Jared also suggests that the sequence: **ifpos *LBL *LBL A ...* will cause SBR A to be called if the display is positive, otherwise "continue". If the sequence following A ends with *rtn, then in the positive case the A sequence is executed twice; in the negative case it is executed once. If the A sequence ends with a HLT it would only be executed once in either case, but the next *rtn encountered would cause a branch to the step following A, unless neutralized by an intervening *rset.

Number Range Notation

The statement: "2 LT INT LT 10^{5}" which appears in the Dynamic NIM game (V1N3p3) means that the input number should be an integer greater than 2 and less than 100,000.

Table Lookup Applications

When a long sequence of functions is to be mechanized, programmers often find it expedient to generate tables, where elements and their ordering determine the functions and their execution ordering, especially if there is no (or no apparent) repetitive pattern to the sequence. In most cases, each table element serves as a pointer to a program-step or register address.

Taking advantage of SR-52 pointer rules (V1N3p1), Alan Trimble (161) was able to apply a table lookup scheme to Dix Fulton's (83) 4 x 4 Determinant program (V1N3p4) to shorten it sufficiently to provide enough room to add a Cramer's Rule solution to four simultaneous equations. The following program is a modification of Alan's program, that speeds up the processing and simplifies inputting. Registers 90-97 contain a lookup table which provides information that specifies the location and sequence of data registers that need to be recalled for determinant calculations. Each of these registers (representing 8 program steps) contains a sequence of 6 pointers to data registers.
Note that 6 straight forward recalls would require 18 program steps. Subroutine C sequences through the 8-register lookup table, moving decimal points to configure register contents for desired pointing. Incidently, I wondered at first how Alan's table loading could work, since some of the table registers contain non-transferable code (registers 90, 92, and 93, since the 8^{th} position in each is a numeral). See V1N2p2 for a discussion of transferability. Fortunately, restructuring by the display is just what is required. For example, the keystrokes: rtx, 1, E, D, C, 7, 6, 5 in Reg 90 (which start at step 160) transfer as: *1/x, 1, *stflg, GTO, LRN, SST, *ifflg, *stflg. This sequence could have been specified to be keyed in in the first place, but the two pseudos: LRN and SST are a nuissance to create.

All the required pointer manipulation, indirect addressing and subroutine calling add significantly to execution time. So, although this program does calculate the determinant of the coefficient matrix (stored in Reg 66), it takes 6 or 7 times as long to run as Dix Fulton's. However, it all fits on one mag card, and there is plenty of room (steps 145-159) to add print instructions. But since there is unrelocatable code at the end, insertions must be made carefully. One safe way is to count them first, then delete a corresponding number of steps in the 145-159 region before making the insertions. Insertions made "above" step 034 will affect the *ifzro absolute branch at steps 069, 070, and 071.

SR-52 Program: Solution To 4 Simultaneous Equations Trimble/Ed

```
Press CLR, then key the 20 equation elements by columns: four X1
coefficients, four X2 coefficients, ... four constants. Follow each entry
with a keyed A. Elements are numbered 0-19, with the next element to be input
shown in each intermediate display. Repeat /:RUN, see Xi :/ i=1,2,3,4

000:  *LBL C 1 SUM 99 (*IND RCL 99 x .01 yx RCL 67) STO 98 *IND RCL 98 *rtn
028:  *LBL D 6 STO 67 89 STO 99 (C X C - C X C)X(C X C - C X C) + 1 INV
061:   SUM 67 RCL 67 INV *ifzro 034 0 = div RCL 66 = *rtn
080:  *LBL B *IND RCL 68 *IND *EXC 69
090:  *LBL A *IND STO 68 1 SUM 68 SUM 69 RCL 68 *rtn 1 STO 66 D STO 66 0
116:   STO 69 E D HLT E D +/- HLT E D HLT E D +/- HLT
133:  *LBL E 16 STO 68 B B B B *rtn         (steps 145 through 159 unused)
160:  rtx 1 E D C 7 6 5 rtx 1 8 8 8 0 0 0 rtx 1 B B B 4 4 4
184:  rtx 1 A *E' 9 3 2 1 rtx 1 6 3 7 D A E rtx 1 1 5 2 9 C *E'
208:  rtx 1 2 7 3 *E' E A rtx 1 5 1 6 C 9 D
```

A Shooting Stars Game (Using Vectored Processing)

Since several members have shown an interest in a game (puzzle) called Shooting Stars (see **Byte** magazine May 1976), here is a mechanization which makes use of vectored processing (see V1N3p5). A Shooting Stars program needs to present an initial 3 X 3 array, then alter it in accordance with predetermined rules, depending on a succession of player's moves. Since the player is limited to 9 possible inputs (the numerals 1 through 9), and since the most processing required (for the number 5) is 9 steps, each input is scaled by a factor of 9 and displaced (by 136) to make it a pointer to the desired program step to begin the processing. Although it might appear that this program can't be very efficient since 24 program steps are "wasted" (steps 143, 144, 152, 153, 159-162, 170, 171, 177-180, 195-198, 206, 207, 213-216 are unused), the efficiency of the vectored processing more than compensates for the wasted space.

Incidently, Shooting Stars does have solutions. Craig Pearce (18) has a winning sequence of 17 shots that works both forwards and backwards. Has anyone succeeded in using fewer shots to win?

SR-52 Program: Shooting Stars (for PC-100 printer) Ed

1. Initialize "Universe": press E; see printed: 111
   181
   111
2. Shoot a star (an 8) by keying a number (1-9) corresponding to the position of a star from the locator:

```
123
456
789
```

At the start, the only star is at position 5. See printed a verification of the shot, followed by a modified Universe which shows that the shot star has been turned into a black hole (a 1), and that surrounding stars and black holes have been "complemented" (stars have become black holes and vice versa) in accordance with the following rules, where * is the shot star, #'s indicate affected stars or black holes, and o's indicate the unaffected objects:

```
* # 0   # * #   0 # *   # 0 0   0 # 0   0 0 #   0 0 0   0 0 0   0 0 0
# # 0   0 0 0   0 # #   * 0 0   # * #   0 0 *   # # 0   0 0 0   0 # #
0 0 0   0 0 0   0 0 0   # 0 0   0 # O   0 0 #   * # 0   # * #   0 # *
```

3. Repeat step 2 until you win: obtain a central black hole surrounded by stars, in which case the winning printed universe is followed by a negative integer indicating how many shots were required. If an attempt is made to shoot a black hole, -1 will be flashed. Recover by pressing CE, and try again (go to step 2).

Program Listing

```
000:  CLR B B B - 2593 - RCL 05 = *ifzro 050 CLR HLT
020:  *LBL B + (C X 100 + C X 10 + C) *prt *rtn
039:  *LBL C 1 SUM 69 *IND RCL 69 *rtn RCL 19 +/- *prt HLT
056:  *LBL *A' *IND RCL 98 - 8 = *rtn
066:  *LBL D STO 98 *A' *ifzro 078 7 + 1 = *IND STO 98 *rtn
085:  *LBL E *CMs 9 STO 00 1 *IND STO 00 *dsz 093 8 STO 05 *rset
106:  *LBL A *pap *prt D *A' *ifzro 134 9 *PROD 98 136 SUM 98 1 SUM 19
       *IND GTO 98
134:  RCL 98 D 1 +/- + = HLT
145:  2 D 4 D 5 D *rset    154:  1 D 3 D *rset     163: 2 D 5 D 6 D *rset
172:  1 D 7 D *rset        181:  2 D 4 D 6 D 8 D *rset 190: 3 D 9 D *rset
199:  4 D 5 D 8 D *rset    208:  7 D 9 D *rset     217: 5 D 6 D 8 D *rset
```

Diagnostic Programs

Mark Stevans (216) points out that TI's Diagnostic Program I (BA 1-18 in the Basic Library) will not produce the intended error codes when malfunctions are detected. The intended results do not occur because no matter what branches are executed, the program always terminates with the same "all is well" sequence. This anomaly can be remedied by replacing the *rtn instructions at steps 035, 050, 065, 084, and 089 with *rset instructions, deleting step 018, and inserting a HLT at step 000. But this program, even as corrected, doesn't really make many significant or critical tests. It would be nice to have a set of comprehensive diagnostic routines that covers all known features of SR-52, SR-56, and PC-100 combinations. So send me your diagnostic routines, and I'll publish the best ones, judged on efficiency and on how well they cover and critically test all the important functions. Fault isolation should be hierarchical, i.e. high level routines identify problems occurring in one or more functions in a large category of functions; low level routines single out specific functions. Blocks of 112 steps (one side of a card) should contain routines of approximately the same hierarchical level.

Programmed Card Write

Several members have reported success at getting cards written under program control. My runaway motor problem (V1N3p4) appears to be intermittent.

Dix Fulton (83) offers the following routine that automatically records 8 data words and retrieves them, with all required code contained on one side of a card: *000: ***LBL A** 8 STO 00 83 STO 01 CLR HLT *IND STO 01 X 1 INV SUM 01 = *dsz 012 INV *read ***LBL B** 83 STO 00 *IND RCL 00 HLT *dsz 037.* Affix write tab to a card and insert in read/write slot. Initialize: press A. Enter 8 numbers, each followed by RUN. Card will be automatically recorded. To demonstrate results, turn calculator off then on, manually read programmed side of card, and press B to retrieve the first recorded number. Press RUN for each of the remaining 7 numbers.

Tips

Efficient Handling of Constants: If a constant is used only once in a program, don't bother to create it and store it, only to have to recall it later. Create it at the needed point. If a constant is used more than once, balance its length against how often it is used. A 3-digit (3 step) constant should be created each time it is needed. A constant requiring more than 8 steps can be more efficiently pre-stored in program memory (provided there is program memory available).

Physical Effects of the Printer on the SR-52: Mike Marquis (205) has found that a mag card left in the top slide when the SR-52 is connected to a powered-on PC-100 can be magnetically altered, especially if left for long periods of time (many hours). Also, printer heat can affect SR-52 performance, notably card-read operation: a problem encountered by at least one member (Bill Kennedy (166)).

Mag Card Care: Phil Sturmfels (49) suspects that sliding mag cards in and out of the viewing window can produce scratches serious enough to cause read or write problems. Does anyone else share this suspicion? (I don't use the viewing window). TI points out in the old SR-52 Owner's Manual (page 176) that scratches and/or oily deposits can cause mag card problems.

Efficient Coordinate Conversion Processing

Many navigation and astronomical problems require spherical trigonometry solutions which in classical form amount to successions of sine, cosine, and tangent manipulations. Since these tend to consume considerable program memory and execution time, it is often productive to try to optimize the algorithms vis-a-vis the machine to be used. Some time ago John Ball (Smithsonian/Harvard Center for Astrophysics) noted that the number of keystrokes required to solve spherical trig problems on the early "scientific" calculators could be significantly reduced by clever use of built-in rectangular/polar conversion functions. I have revised some of his algorithms (intended for RPN machines) for SR-52 mechanization. The following program handles most of the required spherical trig with the *P/R function. I chose this program as an example, since it is also one in which several members have expressed interest. NASA catalogs and bulletins are mailed free of charge to anyone who writes to: Code 512 GSFC, Greenbelt, MD 20771 stating which satellites (up to 20 or so) he would like predictions for.

SR-52 Program: UT/AZ/ALT/Range from NASA Bulletins Ed

```
1.  Key Node Time as HHmm.mm, press *A'
2.  Key Node Longitude as Degrees West, press RUN (flashing pi indicates
    radian mode, in which case switch to degree mode, and continue)
3.  Key satellite latitude as degrees, press A
4.  Key time increment as mm.mm, press B, see event time as HH.mmss
5.  Key Longitude correction as positive degrees, press C
6.  Key satellite height as kilometers, press D
T.  Get azimuths press E, see azimuth in degrees
8.  Get altitude: press RUN, see altitude in degrees
9.  Get Slant Range:  press RUN, see slant range in kilometers
    For new look angle to to step 4; for new node, go to step 2
```

After writing program into memory, but before recording on mag card, in run mode key observer longitude (degrees west), STO 93; key observer latitude (degrees), STO 94. Caution: Insertions or deletions after observer coordinates have been stored should be paired to maintain unrelocatable code at proper steps.

Program Listing

```
000:  *LBL E *fix 2 1 STO 00 RCL 01 *P/R STO 04 RCL 93 - RCL 02 = *P/R
024:   *EXC 04 INV *P/R - RCL 94 = *P/R *EXC 00 *EXC 04 INV *P/R *ifpos
044:   +/- + 360 = *LBL +/- HLT RCL 04 *EXC 00 INV *P/R STO 04 sin div
056:   (RCL 03 div 6370 - RCL 04 cos + 1 = INV tan STO 07 +/- + 90 -
093:   RCL 04 = HLT RCL 04 sin X 6370 div RCL 07 sin = *rtn
114:  *LBL A STO 01 *rtn
120:  *LBL B *fix 4 div 60 + RCL 05 = INV *D.MS *rtn
135:  *LBL C + RCL 06 = STO 02 0 *rtn
147:  *LBL D STO 03 *rtn
153:  *LBL *A' INV *D.MS div 100 = *D.MS STO 05 0 HLT STO 06 *pi sin
173:  *ifzro *pi 0 HLT *LBL *pi *pi INV sin HLT
```

Codes for Mantissa and Exponent Signs

C A Matz (282) points out an error (by implication) in my discussion of sign codes (V1N1p5). Position B (the units place of step 000) detormines mantissa and exponent signs as follows: 0= both positive; 2=mantissa negative, exponent positive; 4=mantissa positive, exponent negative; and 6 both negative.

Streamlined Dynamic NIM

Rick Wenger (235) has done a nice job reorganizing my Dynamic NIM program (V1N3p3) so that it runs considerably faster, and leaves room for added features (including a possible printer version). I suspect that Rick is not alone in wondering why this program works, and asks if there is an algebraic expression for the nth term of the series: 0, 1, 1, 2, 3, 5, ... that the program generates. The series is a Fibonacci sequence (see V1N1p3), and there is indeed a closed-form expression for the nth term: *FN=(((1+SQRT(5))/2)**N-((1-SQRT(5))/2)**N)/SQRT (5)* which I have written as a quasi-FORTRAN statement for typing convenience. For those unfamiliar with FORTRAN, it reads: set FN equal to an expression that is formed as follows: divide the quantity one plus the square root of five by 2, raise this to the Nth power, subtract from this the quantity one minus the square root of five divided by two and raised to the Nth power and divide the result by the square root of five.

SR-52 Program: Dynamic NIM (revised) Ed/Wenger

```
1.  Key starting number, press A
2.  Key your move, press RUN
3.  Repeat stop 2 until there's a winner
```

Program Listing

```
000:  *LBL A STO 01 - 1 = STO 04 RCL 01 *ifzro 164 + RCL 04 div 5 INV *LOG
026:  = INV EE *fix 5 HLT STO 02 +/- *ifpos 049 + RCL 04 = *ifpos 059
049:  pseudo 84 RCL 04 *fix 0 GTO 031 RCL 02 INV SUM 01 RCL 01 *ifzro 174
073:  STO 08 +/- + RCL 02 X 2 = *ifpos 164 0 STO 05 1 STO 06 RCL 06 +
100:  *EXC 05 = STO 06 - RCL 08 = INV *ifpos 096 *ifzro 132 RCL 05
124:  INV SUM 08 GTO 088 RCL 02 X 2 - RCL 08 = *ifpos 157 1 INV SUM 01
151:  X 2 GTO 007 RCL 08 GTO 147 55178.4 *fix 2 HLT pseudo 84 3507.1
181:  *fix 1 HLT
```

Routines

Fractured Display under Program Control: Jared Weinberger (221) has discovered that putting pseudo 31 at program step 223 can get program execution to halt with no decimal point in the display. The sequence 216: ***LBL A** + STO 60 = pseudo 31 when executed with 1.11 in the display halts with all zeros in the display. Perhaps someone can figure out how to get other display format variations without a manually keyed = (V1N2p5).

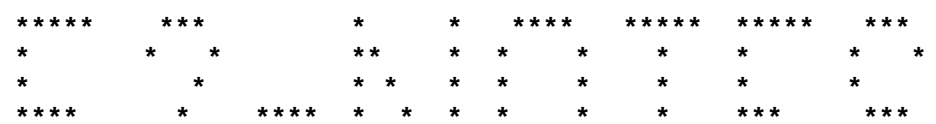
Exponent Extractor

Jared has also devised a routine that displays the exponent (power of ten) as an integer of any non-zero number: ****LBL A*** *(EE div EE 00) *LOG INV EE *rtn*

Calculate Mode Flag Test: Mark Stevans (216) points out that the sequence: *ifflg n 888 (where n=0,1,2,3,4) keyed in RUN mode will result in a flashed display if the designated flag is set. The 888 could, ofcourse, be replaced by any number greater than 223. This method of flag testing would be especially helpful when a large program using flags is being debugged, since it doesn't require any program memory.

Membership Address Changes/Corrections

Make the following substitutions to your membership lists: 3: 890 West End Ave Apt 4C New York, NY 10025. 13: 13723 Sancho Ct Tampa FL 33612. 70: GUDZ, W 88 Baird St Rochester, NY 14621. 75: #312 Willowick Apts College Sta, TX 77840. 81: Box 974 Wrightwood, CA 92397. 189: Box 364 Gardner, Kansas 66030.



Volume 1 Number 5 48/48 October 1976

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

Editorial

With four months and four newsletters behind us, I would like to re-examine Club purpose, goals, and operation.

All things considered, I see no reason to change the original purpose: to get more out of our machines by exchanging ideas. To date, more than 30 members have shared new ideas/provocative questions/useful information that have served as newsletter inputs in one way or another, and I am happy to take this opportunity on behalf of all the membership to thank them. Also, I appreciate the effort taken by many others in sending me material which for one reason or another has not been used. But how about the rest of you? You don't have to be an experienced programmer to discover something new... you mostly need some organized curiosity along with a fair amount of perseverance. I welcome and will publish whole programs that are not destined to join the TI PPX-52 library and which demonstrate, and are accompanied by descriptions of, new mechanizations. But I will continue to give priority to new, clever routines that have broad application. There is still much yet to be explored and discovered: useful pseudo sequences (see the article on pseudos elsewhere in this issue), optimum approaches to a host of combinatorial and numerical analysis problems, new computer programming teaching techniques, extended I/O via the printer interface terminal and card reader, and practical applications of programmed card read/write... just to name a few. I will continue to share my best ideas so long as I have
reason to believe that a reasonable number of other members are doing the same. I don't want to see our Club become mostly a publishersubscriber operation, and I hope most of you feel the same way. Most of what I've just said applies to the SR-56 too, only more so, since I don't expect to provide any input on this machine.

Monetary contributions at the rate of $1.00 per issue continue to provide an adequate operating fund, and I thank those who have contributed more, and who have been sufficiently pleased with newsletters to date to contribute toward publication past the first six issues. I do not intend to issue formal reminders for contribution renewal, but will assume that overdue members are no longer interested, and remove their names from membership and mailing lists. Incidently, I appreciate the fact that members living abroad have volunteered extra contributions when added postage is required.

Let me close this commentary by assuring all of you that I consider my time and effort to be well spent, and it is with pleasant anticipation that I look forward to getting inputs from members who have so far been among the silent majority, as well as to getting more gems from the productive minority.

TI's SR-52 Programming Workbook

TI is giving each PPX-52 member a Programming Workbook which it says normally sells for $4.95. It looks to me as though it might help the novice programmer get started, as it was designed to help SR-52 users who have had little or no prior programming experience. Basic programming concepts are explained in great detail and in very elementary terms. However, I would advise the reader/user to assume that while the sample routines and programs may demonstrate correct mechanizations, they are not necessarily examples of good programming. For instance, it goes without saying that the concept and use of subroutines are important aspects of programming. Yet, a poor example was chosen to illustrate the subroutine section of the workbook. The reader is left with the impression that a one-time calculation, the results of which are used twice, should be made into a subroutine and called twice.

An appendix of advanced programming techniques touches on some of the programming tricks and most of the previously unannounced features already familiar to many of us. Discussion is avowedly brief to discourage the novice from getting hopelessly entangled in exotic schemes. Unfortunately, some critical omissions are likely to cause, not suppress confusion. One such concerns code transferability, and is the omission of the requirement that an octet of program steps may not end in the code for a number if it is to be transferable (see V1N2p2). The workbook implication is that a 0, 2, 4, or 6 in the units place at the top of the octet is sufficient to make the octet transferable.

All things considered, I recommend this workbook to any SR-52 user who finds it difficult to understand the Owner's Manual and/or has done little or no programming, as an aid to getting started. But it is not likely to help anyone to write optimized/complex programs, and it might even be a hindrance.

PPX-52 Status

Word from TI indicates that the first catalog of SR-52 programs will be mailed October 25th to PPX members.

Shooting Stars Solutions

B H Andrews(20), Phil Sturmfels(49), Dwight Kucera(223), C A Matz(282), F W Lehan(300), and E S Molin(307) have all found that they can solve the Shooting Stars puzzle (V1N4p3) in eleven shots. Dwight suggests an analog to Shooting Stars which he refers to as the Big Bang in reverse: the starting "universe" is a black hole surrounded by stars; the end, a star surrounded by black holes. This appears to be an easier puzzle (no doubt helped by the unforced first move), and Dwight has found a solution in five shots.

Shooting Stars Program Addendum

Step 2 of the users instructions for the Shooting Stars program (V1N4p3) should include a "press A" statement that initiates processing.

SR-52 "Crash"

Phil Sturmfels(49) and Chuck Sanford(214) have discovered that a sequence of the form: ***LBL A** **dsz A* ***LBL B** can lead to an apparently unrecoverable "crash" (only the two minus signs show). Some reasonably large integer (99 or 500 or what have you) is first stored in Reg 00. Then the sequence A, HLT, B is keyed manually in RUN mode. If the display flashes, key CE and repeat A, HLT, B until the two minus signs are steadily displayed. The only known way out of this state is to turn the machine off.

Pseudos

A number of members seem to be having difficulty with pseudos, so I will attempt to pull together and expand previous explanations that were apparently not clear to all.

First, let me defend my choice of the word "pseudo" which in the context of SR-52 usage I define as any user-generated instruction code not directly keyable in LRN mode. One member (sorry, I don't recall who) suggests that the word "code" might be a better descriptor, since in computerese, pseudo refers to an instruction which is not directly executable, such as an assembler directive. Well, in this sense, most, if not all SR-52 instructions are pseudos since each starts a sequence of firmware instructions to obtain final results. However, the LRN mode keyable instructions, their associated machine instruction sequences, and execution results were presumably designed into the machine as predictable cause-effect entities and to me qualify under the heading of "SR-52 instructions", as contrasted with what I call "SR-52 pseudos" which set in motion unknown sequences of machine instructions, whose final results may only be partially evident to the user.

But let's press on to the problem of how to create pseudos. I suspect that members who have been unsuccessful so far have also experienced difficulty wading through the register behavior discussions (V1N1p4,p5 and V1N2p1,p2), so I'll avoid technical detail as much as I can, and try a step-by-step approach. If you have not already done so, I recommend writing down a table that shows which program steps are contained in which program memory registers: a reference that you can go to when you need to know that say, step 005 is in Reg 70, or that step 153 is in Reg 89, etc. In other words, write down the complete sequence implied by: 000-007 Reg 70, 008-015 Reg 71, ...216-223 Reg 97. Now, when you need a pseudo somewhere in a program you are keying in, first determine what program memory register it will be stored in. For instance, in the Streamlined Dynamic NIM game (V1N4p6), we find a requirement for a pseudo 84 at step 049. From our table, we see that step 049 is contained in Reg 76. The next thing to do is to create the pseudo as a run mode number which can be stored in the desired program-memory register. In our example, we need an 84 in Reg 76. By following the rules governing register behavior we could place the 84 directly at the desired step (second in our example) within the octet of steps comprising Reg 76 by keying: 1 EE 40 STO 76 8 EE 28 SUM 76. But it is probably easier, and in this case simpler just to key 84 STO 76 GTO 049, and then in LRN mode keying: *del *del *del *del *del *del. In either case any prior Reg 76 contents is destroyed, so the pseudo should be positioned before surrounding code is keyed in. For the NIM program, I recommend creating both pseudo 84s before writing other code. The second one is at step 174 (which is the seventh step in Reg 91), and only requires the sequence: 84 STO 91 GTO 174 LRN *del.

Pseudo **behavior** is something else, and so far as I know has only been superficially explored. Pseudo 84 appears to be a handy one-instruction error-condition producer. But does it do anything else? The sequence: ***LBL A** *pseudo 84 HLT* produces a flashing -1 -99 when executed, but an executed*:* ***LBL A** **pi X pseudo 84 HLT* followed by a manually keyed: CE =, produces fractured digits (see V1N1p3 and V1N2p5). Now, this is just the beginning. What is the effect of preceding and following a pseudo 84 by all possible combinations of all the other "regular" instructions? Add to that all non-duplicating combinations of all 17 pseudos (V1N2p4) with the "regulars", and it is evident that it will be some time before we know all there is to know about pseudo behavior! (Of course, there are probably quite a few interesting sequences of "regulars" that have yet to be discovered, as well.)

Pseudos (con)

There may be an added complication concerning pseudos, as all SR-52s may not respond in the same way when either generating or executing pseudos. Anyone following the suggested steps above to write pseudo 84s in the NIM game who gets different or unworkable results should let me know, stating results obtained, and the machine serial number. Vince Salemme (206) reports difficulty with his machine (#14046) when attempting to fill Reg 70 and 71 with pseudo 26s by storing the number: -2.626262626 -62 in Reg 70 and 71. If you key the sequence: 2.626262626 +/- EE +/- 62 STO 70 STO 71, steps 000-015 should be: 26 06 00 26 26 26 26 26 26 06 00 26 26 26 26 26. Vince claims that step 000 shows up with pseudo 66 instead of 26. According to register behavior rules, a pseudo 66 is positioned directly at step 000 by storing a number like: -1 EE -6 in Reg 70. (The number must have both its mantissa and exponent negative and have a 6 in the units place of the expcnent).

I invite members to experiment systematicaliy with different sequences involving pseudos, and to send me results and conclusions. I'll attempt to put it all together in a future newsletter.

Advanced Programming Techniques (Part II: Table Lookup Optimization)

The method used in the "Solution To 4 Simultaneous Equations" progran (V1N4p2) to squeeze a lot of RCLs into a relatively small amount of program memory may be used to advantage in a broad range of applications. But, as in any attempt at memory-space optimization, it is important not to lose sight of overall objectives. If an efficient practical application is desired, there is no point in saving progren steps when resulting execution time, I/O handling, or error accumulations are unacceptable. And it is even possible that the "overhead" required to control and run optimized routines makes the total number of program steps more than was required in the first place. On the other hand, if the primary objective is to demonstrate a programming technique, inefficiencies don't matter.

For the discussion that follows, let's assume that memory-space optimization is the primary objective. As the V1N4p2 program shows, it is not difficult to pack 6 register-pointers into one table-register. But how about more? If the radix (decimal) point is to be moved two places at a time to present desired pointers, 6 would appear to be the limit. In this case, each pointer can be in the range: 0-99. But suppose the entire range isn't required. In the V1N4p2 program, only registers 00 through 15 need to be pointed to. The capability of pointing to Reg 16 through 99 is unused, and is effectively wasted. Fortunately, this unused range can be traded for more pointers by using another number base. Moving the decimal point two places to the left is accomplished by dividing the number by one hundred. If instead, we move the radix by dividing by sixteen and perform a few base sixteen manipulations, we find it possible to pack up to tan pointers in the 0-15 range into one table-register. Unfortunately, we can't take advantage of the SR-52's ability to act only on two two least significant digits of the integer part of a large positive real (V1N3p1) when it indirectly addresses a register, since the two digits need to be in a base ten (or modified base 10n, n=1,2,...10) system. Instead, we can effectively move the base sixteen radix point to the left in the decimal representation of a base sixteen nunber by successive divisions by sixteen, using the normalized remainder (the decimal remainder is multiplied by 16 and rounded up) each tine as the next pointer, and leaving a successively shrinking integer part for the next division.

But in order to get 10 pointers per table-register this way, we need to carry 13 base ten places, which means using register arithmetic, and integer/ fraction separation within a register. The "overhead" to do all this is significantly greater than that required if only ten places are carried, in which case we can only pack 8 pointers per table-register. So it becomes a trade-off. The following routines show what can be done, and what the overhead "ballpark" is:

```
*LBL A + 16 *PROD 69 0 = SUM 69 *rtn *LBL B RCL 69 div 16 - INV *D.MS INV
*D.MS *fix 0 *D.MS STO 69 = X 16 = *fix 0 *D.MS INV *fix *rtn *LBL C 16 INV
*PROD 69 RCL 69 STO 68 STO 67 1 EE 12 SUM 67 INV SUM 67 RCL 67 STO 69 INV SUM
68 X 16 = INV EE *fix 0 *D.MS *rtn.
```

Routine A packs pointers into Reg 69 and routines B and C unpack them: B for up to 8 pointers, C for 9 or 10. For example, to pack the pointers: 15, 3, 9, 13, 4, 3, 14, 1, 11, 7; key CLR, then each number followed by A. Reg 69 holds the integer: 1046315852215 which represents the ten pointers in packed form. These will only unpack correctly (in reverse order) if routine C is used (just press C for each pointer). Try a string of 8 pointers (in the 0-15 range) and unpack them with routine B. In a practical application, routine A would be used to help write the main program by generating the table-register constants to be permanently stored in program memory. These could either be manually stored as each constant is generated, transfering the contents of Reg 69 to the desired program-memory register, or routine A could be enhanced to do this automatically. Once the main program has been written, routine A is no longer needed, and therefore does not add to the overhead. As written, routines B and C would only unpack Reg 69.

Whichever is to be used in a practical application needs another routine that steps through the permanent table-registers, passing via Reg 69 the required packed data to routine B or C for unpacking.

Let's assume that we have a main program in mind that requires a 100-element lookup table. We will design routine B such that each time it is called, it makes Reg 98 a pointer to a data register in accordance with a pre-determined sequence. We will also assume that the required lookup table has been stored in Reg 82-97 (assuming ten pointers per register). With the following initialization prior to its first call: 10 STO 00 88 STO 99 RCL 88 STO 69, routine E might look like: ***LBL E** C STO 98 *dsz *LBL *LBL *rtn 10 STO 00 1 SUM 99 *IND RCL 99 STO 69 *rtn At each call (from a main program) routine E returns with Reg 98 pointing to the next desired data register. It is, of course, up to the main program to indirectly address Reg 98 after each call to E and do what is desired with the retrieved datum.

Let's take the simple requirement that all the main program needs to do is sum the products of 50 successive pairs of 16 numbers in accordance with a predetermined ordering. If this were to be mechanized by a straight forward succession of 50 groups of steps of the form: RCL ab X RCL cd D, where ***LBL D** = SUM 19 *rtn is the called subroutine D and ab and cd assume values in the range 00-15, over 400 program steps would be required. But if we make use of routine E, then a main program A might look like: ***LBL A** *CMs 10 STO 00 88 STO 99 RCL 88 STO 69 ***LBL *1'** RCL 99 - 98 = *ifzro ***LBL E** *IND RCL 98 STO 18 E *IND RCL 98 X RCL 18 = SUM 19 GTO *1' which although not very efficiently written, gets the whole job done within available program memory. Incidently, the "*LBL E" sequence in routine A does no harm, provided the real subroutine E is nearer the top of program memory than routine A. If the body of routine E were placed following the *LBL E in routine A, a non-0 encountered by the *ifzro test would cause routine E to be executed twice (see V1N4p1).

Table Lookup Optimization (con)

But would we have done better packing only 8 pointers per table-register, and having routine E call B instead of C? Well, we would save 21 steps using routine B, but would need 3 more table-registers, for a net loss of 3 steps: an acceptable alternative since there is enough program memory, and probably better since routine B runs faster than C.

What all this boils down to is that an optimized table lookup scheme should take into account at least three factors: 1) the number of table elements, 2) the range of values assigned to table elements, and 3) the availability of unused program memory that can provide room for speedier code. Thus if the V1N4p2 program were to be "optimized" with denser pointer packing, it would run slower, and there would be no gain, only a loss, as the program already fits on one card. But how about a 4 X 4 matrix inversion, or high-order polynomial curve-fitting programs? Table lookup optimization may be the only way to get them to fit on one card.

Tips

Battery Charger Connecting: Brian Sullivan(247) reports erratic SR-52 behavior if the charger is plugged in after the calculator is turned on. This may be due to connector plug shorting as it is inserted into the socket. It is probably wise not to plug the charger in while the machine is on. Similarly, when using the printer, turn it on first with the calculator locked into place before turning the calculator on.

Exchanging Mag Cards: Members attempting to exchange mag cards recorded on different machines should be aware of at least two potential difficulties: 1) misreads may occur because of different drive/clocking rates, and 2) the fact that a number of SR-52s were manufactured with an implied multiplication following a closing parenthesis. This second problem can be mitigated if programs written utilizing implied multiplication are annotated accordingly.

Mag Card Static Charge: Phil Sturmfels(049) suggests that temporary buildups of static electricity on mag cards may account for their not reading or writing properly on occasion. On one occasion, Phil couldn't get a brand new card to be written on, so shot it with a Zero-Stat static-neutralizing device designed for phonograph records, and some minutes later succeeded in writing on it.

Routines

Jared Weinberger (221) has come through again with another routine, although this time claiming modestly that it might win the "Trivia Award of the Year". I tend to disagree, since it performs a potentially useful service, and the mechanization provides an interesting insight to display manipulations. Jared calls his routine **"Fixed Point Extractor"**, and it lists as follows:

```
*LBL A (10 *LBL *1' (INV EE - 1) + +/- INV *log EE INV EE *ifzro *1' 0) *rtn.
```

When run, this routine returns with an integer in the range 0-9 in the display which represents the display format at the time the routine was invoked. I would only add that the routine appears to work just as well without the inner parentheses. The outer ones are, of course, to protect main-program pending operations if this routine is called as a subroutine.

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 a 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, *$\sqrt{x}$, *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. Ccx (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 ROL SST SST somewhat puzzling. What appeare 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 stops 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 CLRs, 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 GT0 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:

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.

4		
	5	
9		2
	7	
16		2
	9	
25		

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,... < 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 programstored 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 correaponding number of program registers, which would leave 6 (the equivalent of 48 program stops) 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 Weinberger'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.

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 havo no effect. The term "round-up" will be used when all the digits 1-9 cause tho 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 consideration. 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 VIN1p4, 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 0 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 Automatic 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 V1N1p2) is cance]led 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 applied to Reg 03 and 04. Dallas Egbert (384) adds CLR, sin, cos, tan *D.MS, *P/R, y^x, $\sqrt[x]{y}$ 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: ***LBL A** *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 preducing a non-

zero 13th place. This includes the trig, log and exponential functions, 1/x, and x! for x > 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 yx 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 2

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
```

Volume 2 Number 1 48/39 January 1977

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

Puzzling Sequences

It now appears that the topic "Pseudo Behavior" (see V1N5p3) is not big enough to catch all of what is being discovered in the way of strange behavior, so I will consider in this space reported combinations of anything manually, SST, *bst, or program executable that produces a new result. I'll explain what I can, and invite corrections, enhancements, new explanations, etc from the membership at large. Henceforth, pseudos will be designated with a p followed by the synthetically creatable 2-digit code (p31 means pseudo 31); keyboard-creatable instructions will be designated by the appropriate mnemonic (*INV' is op-code 27).

A *LBL Peculiarity (52): Michael Brown (128) and Dallas Egbert (384) have found that if in RUN mode keying LRN is preceded by *LBL, the program step pointer is advanced one step. Expanding this a little further, I find that *LBL SST LRN advances the pointer 2 steps. It also appears that the pointer moves one step if the *LBL and the LRN are separated by keyed numerals (which leave a soft display). However, *LBL CLR LRN does not move the pointer. Then, if the pointer is positioned at step 223 with *LBL LRN executed manually in RUN MODE, apparently nothing happens (a second LRN reveals step 223 in LRN mode). This is probably related to what happens when you execute an instruction at stop 223 with SST: it takes 2 successive LRNs to get into LRN mode. Incidently, *LBL p31 appears to behave like any other label under program control.

***INV' Acting Like p21:** Dallas has been experimenting with sequences involving *INV' that make it appear to behave like the p21 pseudo (2nd). It should be noted that while p21 as a separate step doesn't always behave like a "2nd" when merged with an instruction, which it shifts, in the cases Dallas tried, it does: put *INV' at step 223 (99 for SR-56), then in RUN mode (at step 223 (99)) key SST, and it will take 3 successive LRNs to get a switch to LRN mode; write beginning at step 000: ***LBL A** *INV' p31, press A, press any non-edit button, then backstep to find the code for the shifted operation of the previously keyed button; press A (in RUN mode), then SST, and find that it has executed as *bst. Substitute p21 for *INV' in the above sequences, and the results should be the same. This observed behavior suggests that program or SST execution of either *INV' or p21 can produce a viable detached pending shift which can affect manually keyed operations in either RUN or LRN modes. In the 3 LRNs example, the first LRN is executed as *IND (*f(n) for SR-56) and it takes 2 more following a step

223 (99) SST to get into LRN mode (see "A *LBL Peculiarity" above). Speaking of abel peculiarities, unlike *INV', p21 when addressed as a label causes a reset with error condition.

SST'd SST (52): In the course of his examining *INV' behavior, Dallas found that execution of p71 (SST) by a manually keyed SST causes the program pointer to skip over (or perhaps execute) the p71 and to execute the next instruction. This effect appears to hold for any number of successive p71s. (In RUN mode key: 1.009717171 STO 70 GTO 003 SST, and see the trailing 9 that was executed after the program pointer skipped over (executed?) the 3 p71s)

.001 Dim Display: Chuck Sanford (214) finds that with the SR-52 plugged into the PC-100, the .0000001 dimming (V1N3p6) does not occur, but that .001 causes dimming. Thus there appears to be a power supply dependency for this display phenomenon (which holds for the SR-56 also).

Alphabetized Membership List

Several members with access to general purpose computer systems have offered to prepare alphabetized membership lists, and I greatly appreciate the volunteered help. The most attractive came from Michael Brown (128) who managed to get an IBM 1130 system to type the output directly on to mimeo masters. The resulting list is included with this issue of 52-NOTES that goes to members. Many thanks Mike for providing a valuable service that minimizes the publishing effort. Incidently, I will note here that I intend to continue printing 52-NOTES via my home-based mimeograph until such time as circulation should become too large to handle. I like to be able to get ephemeral topics in on short notice, and will assume that a majority of the membership prefers topical timeliness to unblemished printing (until I get word to the contrary).

TI Notes

Machine Anomalies: To date, TI has issued one "addendum" for the SR-56; none for the SR-52. The one for the SR-56 was reportedly included with the first retailed unit, but apparently Sandy Greenfarb (200) didn't get one. The jist is: m + or - 0 y^x n=mD12 and m + or xrty n = nD11 (the xrty part courtesy of Sandy, who discovered both effects while debugging a practical program).

Games PAC: The Games bonus offered in the December **Scientific American** TI advertisement to buyers of new SR-52s will be available as a regular PAC 15 January 1977.

PPX-52 Publications: The first issue of the PPX-52 Newsletter was due to be published the last week of December. The next catalog is due about February.

Machine Exchange: TI now has a number of centers at which users may exchange defective machines (in warranty) for new ones. Call Consumer Relations for the one nearest you.

Corrections

SR-52 Timer Instructions (V1N7p6**):** Step 8 should reads "For new timing exercise, go to step 2".

Routine B (V1N5p5**):** The second *fix 0 is superfluous.

Register Behavior: Part III Overflow And Underflow

(Herewith the article that I mistakenly said last month would be in V1N7.)

The SR-52 and SR-56 Owner's Manuals tell us that both machines were designed to be in an error state any time either display or register arithmetic causes (or tries to cause) the creation of a number whose absolute value is smaller than 1D-99 or larger than 9.999999999D99, and, for the SR-52 that *"... When direct register arithmetic results in underflow or overflow of that register, the error condition remains until the contents of that register are changed."* (see page 183 of the SR-52 manual). As it turns out, it is possible to create a number larger than 9.999999999D99 without triggering an error state, and the presence of an error-state producing overflow or underflow value in a register does not of itself cause (or continue to cause) an error state. If the 11th, 12th, and 13th places are 499 or less, then there is no overflow, and if register arithmetic causes an overflow, CE executed either manually or under program control appears to turn off the error state. A subsequent recall of an out-of-bounds number would, of course, re-invoke the error state. And then, it's even possible for an SR-52 program-memory register to contain a sequence of op-codes which if interpreted as data would constitute an overflow or underflow value. For example, in RUN mode key: 9.999999999 EE 99 STO 70 9.99 EE 89 SUM 70. This should not cause the machine to go into an error state. Yet, if RCL 70 is keyed, an overflow error condition is created. Or, if (SR-52 only) the sequence: +/- *PROD 0 0 0 0 0 0 is written starting at step 000, then RCL 70 creates an underflow error condition.

In the discussion that follows, let's examine register contents as they appear in op-code format, using the designators: AB, CD, ... OP (see V1N1p5), determine the specific characteristics that cause overflow and underflow, and consider the concepts of "conditional" and "absolute" overflow. Key the sequence: 9.999999999 EE 99 STO 70 4.99 EE 89 SUM 70. Reg 70 should now contain the number 9.99999999499D99. Now RCL 70. All 9's should show, but without error condition. Now key: 1 EE 87 SUM 70. No error condition; but after RCL 70 there is. Steps 000-007 should look like: 90 09 50 99 99 99 99 99 which represent the number 9.99999999500D99. Note that a one was added to the 13th place changing positions CD from 99 to 09, and EF from 49 to 50. Register arithmetic occurred properly, i.e.: 9.999999999499D99 + 1D87 = 9.999999999500D99. Yet the resulting sum is regarded as too large when displayed, because of the attempted rounding to ten significant figures. I suggest that Reg 70 is now conditionally overflowed; it contains a number which can be operated upon normally by register arithmetic, but which when displayed creates an overflow error state. This is true for all numbers in the range from 9.999999999500D99 to 9.999999999999D99 inclusive. The setting of the overflow error state during register arithmetic only occurs when the result would be larger than 9.999999999999D99, in which case this conditional overflow number is left in the register.

Now let's see what happens with underflow. In RUN mode key: 1 EE +/- 99 STO 70. Steps 000-007 should look like: 94 09 00 00 00 00 00 10. This is the smallest number the machine can hold (putting anything but zeros in the 11th, 12th or 13th places (positions E, F, C) only makes the number larger). Now try to make the number smaller by keying: .1 *PROD 70. No error condition. Analogous to overflow, the number left in the register is the smallest possible, but unlike overflow, the register is not "conditionally" underflowed; i.e. bringing it into the display does not cause an error condition.

Now let's see what happens when we artificially create a "number" that is too small to recognize as a datum (SR-52 only). Cycle the on-off switch, then in LRN mode at step 001 key *E'. Now in RUN mode key RCL 70, and see 1D-12. The machine moved the one at position C 12 positions down to position 0 in order to format it acceptably for display (see V1N2p2) and compensated for this by the D-12. Now, in LRN mode beginning at step 000 key: +/- *D' 0 0 0 0 0 0. From what happened in the preceding example, we might expect the machine to interpret this "number" as 1D-111 which is, ofcourse, too small to display. Indeed, if in RUN mode you RCL 70, you will get a flashed 1D-99. The smallest artificially creatable legal number would look like: 84 09 00 00 00 00 00 01 which transfers as 1D-99. Apparently artificially created too-small numbers cannot be expanded via register arithmetic to become "legal". If 1D-100 (represented in steps 000-007 by 94 09 00 00 00 00 00 01) is multiplied by 10 (via register arithmetic), the result is 1D-98 (instead of the correct 1D-99) and an error state created. In fact, register arithmetic performed on any artificially created underflow number, treats that number as 1D-99 before operating on it.

Some of the implications of all this are: 1) Either conditionally or absolutely overflowed registers will respond normally to register arithmetic, provided absolute overflow does not result, 2) RCL of either conditionally or absolutely overflowed registers creates an error condition, 3) there is no conditional underflow, 4) RCL of an underflowed register does not create an error condition, and 5) register contents that are interpreted as less than 1D-99 are treated as 1D-99 during either register arithmetic or when recalled, and an error condition is created in either case.

The Matrix Challenge (52)

Some of us have found it both challenging and rewarding to write programs that solve common matrix problems. Much of the challenge lies in how to determine the best approach, as well as how best to mechanize the chosen one. I first confronted the SR-52 4 x 4 determinant problem a year ago, and thought that a fairly straight forward FORTRAN algorithm for a Gaussian Elimination method would work well. I managed to get a working program to fit on one card (which could also be modified to get a 5 X 5 determinant) but it wouldn't handle column exchange when zeros were produced on the diagonal. So I modified the main program so that it would automatically read a second card that would exchange columns if required (4 X 4 only). It was great fun working out multiple pointer manipulations and program overlays, but the program didn't perform nearly as well as Dix Fulton's straight forward approach (V1N3p4). I haven't yet come across a program superior to Dix's for just calculating a 4 X 4 determinant, but it is tempting to try to squeeze in other matrix operations all on one card. Using a table lookup scheme to save space (V1N4p2) Alan Trimble tacked on a solution to 4 simultaneous equations that works well, but appears to be surpassed by Rick Wenger's (235) program (below) which runs faster and requires fewer steps, mechanized with an extensive network of subroutines. Incidently, there is a clever flag shortcut in Rick's programs a call to the undefined D function sets the "flag" and *iferr provides the test. (It's just too bad if a real error shows up!)

For a matrix inverse solution, it appears that no matter what approach you take, there are a lot of computations involved... more than can be fit into a one-card SR-52 program, without clever manipulations tricks.

I tried a table lookup optimization (V1N5p4,5,6) approach, but still couldn't squeeze it all in. However, there is at least one way that works, as Barbara Osofsky (420) shows in the program that follows (modified with a few I/O, throughput, and space-saving enhancements, including Rick's flag shortcut). A 92-step subroutine does all the arithmetic; the other 132 steps handle input, output and the required register manipulation to make 20 calls to subroutine *A'. Although this program was written to be used with the printer, it can be used without the printer by substituting HLTs for the *prts at steps 082 and 191. Barbara claims to be close to a one-card 5 X 5 inversion... good luck! Then, of course, it would be nice to combine the determinant, inverse, and equations solutions all on one card. The four equations constants can be input following the matrix elements in Barbara's program, but Cramer's Rule column exchanges would need to be done manually. Perhaps someone can combine Rick's and Barbara's programs into one!

SR-52 Program: 4 Simultaneous Equations Rick Wenger (235)

User Instructions: Same as for the V1N4p2 **program.**

Program Listing

```
000:  *LBL E STO 68 *IND RCL 68 *rtn *LBL *A' E X RCL 15 - RCL 14 X *rtn
023:  *LBL *B' E ) X ( *rtn *LBL *C' E X *rtn *LBL *D' E - *rtn
040:  *LBL *E' E ) + ( *rtn *LBL C E X RCL 13 - RCL 12 X *rtn
      *LBL B *IND RCL 68 *IND *EXC 69 *LBL A *IND E 1 SUM 68 SUM 69 RCL 68
      *rtn
085:  0 STO 69 D (10 *A' 11 *B' 0 *C' 5 *D' 1 *C' 4 *E' 6 *A' 7 *B' 1 *C'
111:  8 *D' 0 *C' 9 *E' 2 *A' 3 *B' 4 *C' 9 *D' 5 *C' 8 *E' 8 C 9 *B'
133:  2 *C' 7 *D' 3 *C' 6 *E' 4 C 5 *B' 3 *C' 10 *D' 2 *C' 11 *E' 0 C
157:  1 *B' 6 *C' 11 *D' 7 *C' 10 *E' 0 = *iferr 197 div RCL 66 = HLT
181:  1 +/- *PROD 66 16 E B B B B GTO 090 STO 66 CE GTO 186
```

SR-52 Program: 4X4 Determinant and Inverse B.Osofsky (420)/Ed

User Instructions

1. Initialize with CLR
2. Key first element, press E; see 1 displayed, and printed confirmation
3. Key ith element, press E or RUN; see i displayed and printed confirmation. Repeat for i = 2, 3, ... 16 with row-wise catenation
4. Press A, and get printed the determinant, followed by the 16 inverse elements grouped by rows. Program ends with -1 displayed. For a new problem, go to step 1.

Program Listing

```
000:  *LBL *A' RCL 05 X (RCL 10 X RCL 15 - RCL 14 X RCL 11 ) + RCL 06 x
028:  (RCL 11 X RCL 13 - RCL 15 X RCL 09 ) + RCL 07 X (RCL 09 X RCL 14 -
059:  RCL 13 X RCL 10 = div RCL 99 +/- STO 99 +/- = *ifflg 0 083 *prt
083:  X RCL 00 = SUM 69 *rtn *LBL C + 3 STO 98 = STO 68 *IND RCL 98
107:  *IND *EXC 68 *IND STO 98 4 *iferr 121 1 +/- SUM 68 SUM 98 RCL 98
131:  *ifpos 103 CE *rtn *LBL B *A' 4 C *A' 8 C *A' 12 C *A' 12 C 8 C
155:  4 C *rtn *LBL *D' *pap D + 1 +/- *PROD 99 12 GTO 096 *LBL A *pap
177:  *stflg 0 1 STO 99 CLR B INV *stflg 0 RCL 69 *prt *pap STO 99 B 1
198:  *D' B 2 *D' B 3 *D' B *rtn *LBL E *IND STO 69 *prt 1 SUM 69
218:  RCL 69 HLT GTO E
```

Forum

PC-100 Hardware Problems: Gerald Donnelly (203) has had several printer problems, and wonders what is the best way to get satisfaction. Since it now appears that many of the early PC-100s were poorly designed or fabricated, my advice is to keep exchanging units until you get a good one (Gerald and I are already on our second replacements). Just make sure you find all the faults before your warranty expires! Gerald suggests that it might be helpful to the membership to tally incidences of hardware problems. Let me know yours, and I'll pass the info along in a future newsletter.

Duplicate PPX-52 Programs: Barbara Osofsky (420) brings up the matter of PPX-52 possibly rejecting a program because it produces the same results as one already in the library, even though it may be superior in some way. She suggests that Club members could share their PPX-52-rejected but superior programs. This looks like a good idea, and I invite members who wish to participate to send me an abstract of a rejected superior program, the ways it is superior to a corresponding PPX-52 program, and to be willing to provide copies to members who send a SASE and money (state how much you require) to cover copying costs. I will publish such declarations in future newsletters.

SR-52 Pause Function: Don Williams (29) and Shuichi Takahashi (422) would like to know if anyone has been able to create an SR-56-type pause function for the SR-52.

Book Review: Applied Mathematical Physics with Programmable Pocket Calculators Robert N Eisberg (McGraw-Hill 1976, 176 pages)

The capabilities and availability of the SR-56 and HP-25 class of personal programinables led physics professor Eisberg to consider a new approach to introducing selected topics in physics at the college level. By mechanizing elementary numerical solutions to differentiation, inte- gration, and differential equations for both the SR-56 and HP-25, Eisberg introduces physics problems to the novice which would be either too difficult or just plain unsolvable using analytic techniques. Topics range over linear and central-force motion with friction taken into account; mechanical and quantum theory oscillators with resonance, damping, and coupling; and random processes. The programs appear to work, although most could be readily optimized for more efficient I/O. And since most output is destined to be plotted, incorporation of the PC-100 printer with the SR-56 programs would be a significant enhancement. This looks like a good book for layman personal programmable users as well as physics instructors and students.

Correction: V2N1p4 (bottom)

The sequence: 1 EE +/- 99 STO 70 .1 *PROD 70 does create an error condition; subsequent RCL 70 does not.

Membership Address Change

Make the following change to your membership list: #281: #18 (not #17).

Volume 2 Number 2 48/39 February 1977

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

Tips

Clever Uses of INV P/R: Dix Fulton (83) suggests the following sequences to handle raising a negative number (in Reg 01) to an integer power (in Reg 02): for the SR-52: RCL 01 STO 00 0 INV *P/R X RCL 02 = cos X RCL 00 y^x RCL 02 =; for SR-56: RCL 1 x:t 0 *f(n) R-P X RCL 2 = cos X x:t y^x RCL 2 =. Jared Weinberger (221) notes that: 0 INV *P/R for the SR-52 or 0 *f(n) R-P for the SR-56 replaces the contents of Reg 00 (t) with its absolute value. For a practical application of these facets of INV *P/R see "Short, fast quadratic solutions" in the Routines column below.

More on Charger Connection: Larry Mayhew (145) reports that John Allen's charger connection method (V1N7p4) appears to cause execution speed to slow slightly (he had to change the constants in his timer program (V1N7p6) to around 9600... a change of 1-2%). This condition appears to hold until the machine is turned off. Subsequent "normal" turn-ons apparently produce normal execution rates. Anyone care to venture a guess as to what's going on?

A Decrement-Only dsz (52): Larry has found the sequence ... *dsz 0 op ... (where op is any non-numeral instruction) to work whenever Reg 00 is to be decremented without a zero-test for transfer.

1/0=0: Larry further notes that in cases where reciprocals of sequenced numbers are calculated and it is desired to have 1/0=0, the sequence: n y^x 1 +/- = will work, where $n \geq 0$. However, results are not always exact, and rounding may be necessary if integer results are required.

Trig Function Anomalies: In the course of trying to get a practical program to work, Larry found that there are normal trig sequences which can hang up and produce an error condition. There appear to be critical numbers close to 90° (pi/2 in radian mode) which when operated upon by INV sin produce numbers slightly larger than one, and other critical numbers close to zero which when operated upon by INV cos also produce such numbers (SR-56 too). Unfortunately, some of these critical numbers can be created by the sin and cos functions themselves. The sequence (in degree mode): 0 INV cos sin INV sin produces an error condition, since 0 INV cos sin produces 89.99999999987 and the machine sine of that is 1.000000000001. The worst case I have found so far is to take the Sine of 89.99999999540 which produces 1.000000000004. So the moral of the story is: if in doubt, round!

Tips (con)

More on + STO 6O (52): Chuck Sanford (214) and Dallas Egbert (384) note that the sequence: + STO 60 = leaves a pending INV, and Chuck points out that + STO 60 = EE is a short manual method of displaying the 4th through 13th digits (of numbers greater than one). The EE executes as INV EE.

The Printed "?" as an Identifier (PC-100): Sandy Greenfarb (200) has found that purposely creating an error condition prior to a *prt is a handy way to mark selected outputs, since the printed "?" is easy to spot as well as to create.

Step 223 (99) *list: executes as a normal HLT (R/S) when run with the printer; otherwise a run-past-last-step error condition is created.

Optimized Hardware Interrupts: Jared Weinberger (221) points out that the approach suggested in V1N2p4 (top) requires more execution time than necessary, since advantage can be taken of the fact that the machine executes the sine function quicker if the result is zero than otherwise. If the sequence is rearranged as follows: ... **pi sin *ifzro *pi HLT* ****LBL *pi*** *...* then uninterrupted loop execution occurs in radian mode. SR-56 mechanization needs to take into account the fact that the choice of angle modes during program execution is between degrees and grads. The sequence starting at step 00: 90 cos *x=t 07 R/S ... RST executes uninterrupted when in degree mode and halts at the H/S when in grad mode. (The t register must be zeroed prior to execution)

Re-softening a Hardened Display: Dallas Egbert (384) notes that the sequence EE INV EE softens a hardened display to the extent that successive keyed numerals are positioned at the LSD of the mantissa. For example, key: 1.23 = EE INV EE 45 and you get 1.2345. Dallas also notes that any hardened number followed by EE INV EE CE can be fully restored via Ed Haas' routine (V1N6p3). However, the EE rounds the mantissa to ten places.

Routines

Short Fibonacci Sequence (56): Paralleling the SR-52 routine (V1N1p3), Sandy Greenfarb (200) has devised a short SR-56 Fibonacci sequencer which, slightly modified, lists starting at step 00 as: + *pause x:t RST; (x:t means x exchange t). In run mode key: CLR x:t 1 RST R/S, and see each successive Fibonacci number pause-displayed. (Replace the *pause with a *Prt for printed output via the PC-100).

Short, Fast Quadratic Solutions: A couple of years ago, John Herro (HP-65 Users Club member #47) found a clever use of the rectangular to and from polar conversion functions (see Tips above) in developing an efficient algorithm for calculating both real and complex roots of quadratic equations with real coefficients. The following routines are my mechanizations of John's algorithm: for the SR-52:

```
*LBL A STO 01 *1/x X HLT div 2 = STO 02 *x² - HLT div RCL 01 = STO 00 0 INV
*P/R div 2 = *EXC 00 *√x *EXC 00 *P/R HLT RCL 00 - RCL 02 = HLT RCL 02 +/- RCL
00 = *rtn;
```

for the SR-56: starting at step 00:

```
  STO 1 *1/x X R/S div 2 = STO 2 x² - R/S div RCL 1 = x:t 0 *f(n) R-P div 2 =
x:t *√x x:t *f(n) P-R R/S x:t + (CE - RCL 2) R/S RCL 2 = +/- R/S.
```

To run them (SR-56 instructions in parentheses): Key a, press A (RST, R/S); key b, RUN (R/S); key c, RUN (R/S); see imaginary part of root displayed; key RUN (R/S) and see real part of root one; key RUN (R/S) and see real part of root two. a, b, c are the coefficients of a quadratic in the form $ax^2 + bx + c = 0$.

Analysis of a Super Program

Barbara Osofsky (420) has indeed produced a one-card program that not only calculates the determinant and inverse of a 5 X 5 matrix, but also does a 4 X 4 determinant and solves a system of four simultaneous equations. While many of you may be wearying of matrix programs, I think you will find the time taken to see how this one is structured to be well spent, since many constructs are applicable to other how-to-get-it-all-to-fit situations, and the analysis that follows is concerned with programming techniques, not applied matrix theory. As it lists below, Barbara's program was modified to reduce the amount of manual keying required, and a few labels and registers were rearranged for I/O ease.

Let's begin by looking at the input routine, beginning at step 138. At first glance it looks straight forward enough, except for the error test and the extra summing. But note that routines D', D and A' all run right through it! Thus routine A is used each time D', D or A' is called (which is quite a few) besides for inputting data. Routine B' does most of the arithmetic, and is called 60 times for the 5 X 5 determinant and 300 times more for the inverse. Note that routine B' is written "in-line", that is, it doesn't call any subroutines itself. This is necessary because it itself is a third-level call (by E to C' to B'), and it would run slower, even if it could be shortened with subroutine calls. Note that space is saved by running it into E'. Routine D' heads up the D' D A' A chain, and calls a part of itself four times before running directly through it (the A' A part). The only conditional branch occurs as an *iferr flag at step 145 (see V2N1p4). The flag is set with a user call to the undefined label C, and reset with the CE at step 166. An important output (the 5 X 5 determinant) is created just prior to execution of step 180; yet without parentheses it is pushed into Reg 60 with a + operator and safely preserved during the execution of some 130 succeeding steps, until it finally emerges at step 191 for disposition. This is possible because no intervening display arithmetic occurs.

Throughout, it is careful structuring and ordering of subroutines that saves the most space, and makes it possible for the user to accomplish so much with a one-card program. The program itself had to be fit within 200 steps, since 3 program storage registers are used for data.

SR-52 Program: 5 X 5 Determinant and Inverse; 4 x 4 Determinant and 4 Simultaneous Equations (with PC-100) Barbara Osofsky/Ed

5 X 5 User Instructions:

1. Initialize: 95 STO 66
2. Input the 25 elements (e_i) with column-wise catenation: key e_i, press A, see 1; repeat for i=1,2, ... 25.
3. Get the Determinant: press B then E; see determinant displayed after about 2 minutes.
4. Ignore 4 lines of print: On printer press ADV, tear off tape, press PRINT, ADV.
5. Get Inverse: Press B then E; first column printed, see 1 displayed.
6. Key next column number (r), press RUN, C then D then E; rth column printed, 1 displayed; repeat for r=2,3,4,5.

5 x 5 User Instructions (con)

Notes:

1. If elements are input with row-wise catenation, inverse elements are printed by rows.
2. Before step 3 execution, the number 1 must be displayed; before step 5 execution, the determinant must be displayed.
3. At step 6, do not cancel error condition caused by pressing C; routine D ends by flashing 5.

4 x 4 User Instructions:

1. Initialize: 95 STO 66 1 A
2. Input the constant column (c_i): Key c_i, press A; see 1; repeat for i=1,2,3,4.
3. Input coefficients (a_{ij}) by column: Key a_{ij}, press A; see 1; repeat for i (row) =1,2,3,4,5 and j (column) =1,2,3,4 where a_{ij}=0 are four dummy elements that must head each column. a_{54} is the last coefficient, and should be in Reg 19 (following RCL 19, replace the 1 in the display before proceeding).
4. Get Determinant: Key B then *C'; determinant is printed; on printer key ADV.
5. Key 0 = +/- B then E; see printed: -1, x_1, x_2, x_3, x_4; -1 displayed. Original input ordering of elements is returned (Reg 95-19).

Program Listing:

```
000:  *LBL *B' STO 68 (*IND RCL 66 x *IND RCL 68 - 1 SUM 66 +/- SUM 68
024:  *IND RCL 66 X *IND RCL 68) X *LBL *E' 1 SUM 66 *rtn *LBL *C' (8
046:  STO 66 4 *B' *E' 17 *B' +/- + 19 *B' STO 66 7 *B' -14 *B' *E' 17
072:  *B' + 19 *B' *E' 2 *B' + 9 *B' *E' 2 *B' - 4 *B' 6 STO 66 12 *B')
097:  div RCL 67 B X *prt RCL 95 + *rtn *LBL *D' 10 INV SUM 69 *LBL D
119:  95 STO 66 *A' *A' *A' *A' *LBL *A' *IND RCL 66 *IND *EXC 69
138:  *LBL A *IND STO 66 5 *iferr 150 1 SUM 69 SUM 66 *rtn *LBL B +/-
160:  STO 67 *rtn *LBL E CE 0 STO 69 *C' D *C' D *C' D *C' D *C' 5
181:  SUM 69 *D' *D' *D' *D' 0 = *pap HLT + 94 = STO 69 HLT
```

Magazine Review

Members have asked for information concerning personal programming-related periodicals. Herewith a start. I invite reviews of other related periodicals from the membership.

Byte: (the small systems journal) is aimed at the computer hobbyist, and appears to be a fairly respectable commercial operation (total paid circulation 30 Sep 76 of 62,280). Published monthly by Virginia and Manfred Peschke at 70 Main St, Feterborough, NH 03458 at $12 per year, you'll have to scrounge for back issues. Topics range over both hardware and software aspects of building and using the class of machines centered on the 8080 and 6800 type microprocessors, although personal programmable calculators have occasionally been the subject of articles and letters.

Popular Computing: is published monthly by Fred Gruenberger (27) at Box 272 Calabasas, CA 91302; Jan 77 is the 46th issue. Each issue runs 20 unreduced typewritten 8½x11 pages, with almost no advertising. Topics generally concern numerical problems and puzzles whose solutions are apt to require the use of large computing systems. However, there are problems which can be productively explored with personal programmables. On occasion computer-related philosophical topics are aired. Subscription is $20.50 per year; back issues $2.50 each.

A Fractured Digits, All Flags, Game Application (52)

Alan Charborneau (306) found that dice rolling for the poker-like game of Yahtzee could be mechanized for the SR-52 in such a way as to take advantage of display variations (V1N2p5), as well as to pose a selection requirement nicely met by using all 5 flags. With a few optimization revisions, Alan's program appears below. Routine D' is called by the main program (which starts at step 038) each time a new die roll is required. Dice values are temporarily stored and selectively changed, then integrated with a skeleton string in Reg 06 for Reg 60 manipulation to produce the 5-dice display. The flags "freeze" selected dice for partial rolls. The required flag manipulations spotlight how nice it would be to have a function that only resets all flags (not do everything *rset does). Perhaps there is a "puzzling sequence" (V2N1p1) that would do the trick. p73 is sort of an inverse of what is required since it causes a branch to step 000, but does not reset flags.

SR-52 Program: Yahtzee — Alan Charbonneau (306)/Ed

User Instructions:

1. Key random number seed: 0 < seed < 1, press *A'; see 9D99; press =, see first roll of 5 dice: d_1-d_2-d_3-d_4-d_5
2. (Optional) Save dice by pressing one or all of A, B, C, D, E for corresponding dice positions
3. Press RUN to roll remaining dice (if step 2 is omitted, all 5 dice are rolled); see 9D99; press =, see next dice configuration. Go to step 2 for dice save, or repeat step 3 for new roll.

Program Listing:

```
000:  *LBL *D' RCL 98 X *pi = STO 98 - .5 = *fix 0 *D.MS INV SUM 98
022:  RCL 98 X 5.5 + 1 = *D.MS *IND STO 00 *rtn 5 STO 00 *ifflg 0 048
047:  *D' INV *stflg 0 *dsz 0 *ifflg 1 059 *D' INV *stflg 1 *dsz 0
064:  *ifflg 2 070 *D' INV *stflg 2 *dsz 0 *ifflg 3 081 *D' INV *stflg 3
084:  *dsz 0 *ifflg 4 092 *D' INV *stflg 4 1 EE 95 STO 99 5 STO 0O
106:  RCL 97 STO 06 *IND RCL 00 X RCL 99 = SUM 06 100 INV *PROD 99
131:  *dsz 112 RCL 96 + STO 60 RCL 06 HLT CLR GTO 038 *LBL *A' STO 98
156:  CLR GTO 038 *LBL A CLR *stf]g 0 GTO 037 *LBL B CLR *stflg 1
175:  GTO 037 *LBL C CLR *stflg 2 GTO 037 *LBL D CLR *stflg 3 GTO 037
197:  *LBL E CLR *stflg 4 GTO 037 0 0 *pap cos 6666 . *pap *ifzro
217:  90000 *ifzro *ifzro
```

Forum

Sick Machine Exchange: Robert Hausafus (431) suggests calling your nearest TI machine exchange center before taking your machine in; they may not have the unit you need in stock, in which case ask them to order one for you. Robert's problem with a PC-100 may be a common one: inoperative character dots.

Applications Topics: Thomas Watkins (347) suggests that each month 52-NOTES focus on one applications topic (games, surveying, aviation/navigation, physics/chemistry, etc), emphasizing routines that maximize user efficiency and ease. I invite members to send me their algorithms, routines, approaches; or a description of problems encountered trying to get a particular applications program to work. As info builds sufficiently on any one topic, I'll publish the material in 52-NOTES.

SR-56 Program Exchange Update (See V1N6p6**)**

Dave Johnston's 1 Feb 77 Catalog lists 46 programs on math, statistics, physics, games, and misc. Members wishing to have access to SR-56 programs covering technical disciplines or specific problems not already addressed are invited to speak up. Send your requests to both Dave and me.

Miscellany

Corrections to TI Software: Since most applications library and PPX-52 programs address specialized topics, corrections and revisions will not be given space in 52-NOTES. TI is the appropriate focal point.

Display Arithmetic Modification: Machines shipped after late October 1976 were manufactured with a new chip that automatically truncates the 13th place of a display operand just prior to its interaction with a Reg 60 operand. Display values operated upon singly and data register contents are not affected, and retain up to 13 places in the usual manner. A quick test to see how your machine behaves is to key: 3 *1/x - STO (CE for SR-56) =. A resulting zero signifies the new chip; -3D-13 the old. A word of caution: While most programs written for old-machine execution should work on the new ones (upward compatibility), the reverse may no t be true since some rounding or undesired register arithmetic can be dispensed with when programming the new machines.

SR-52 Successor: Unofficial sources point to the introduction of a new machine in ten weeks or so that will probably be marketed as SR-52 II. It is likely to combine SR-52 and SR-56 features and fill in some or all of the "missing" SR-52 registers (20-59). Printer compatibility will probably be limited to the PC-100A (which has a 3-position calculator selector switch).

Program Listing Conventions: Although some have suggested dropping the * shift indicator, I'm going to keep it for program listings, as it's too easy to forget the shift key without the reminder when punching a new program from a listing. When functions are referred to in the text of an article, the * will be omitted. User instructions designed for both SR-52 and SR-56 use will show SR-56 instructions in parentheses following the SR-52 instructions they replace.

A New Facet to Registers 20-59: It turns out that these 40 "non-registers" point to Reg 00 or step 000 when invoked with *IND. For example key: 999 STO 00 CLR *IND RCL 45; see 999. Or, key: GTO 123 *IND GTO 59 LRN and see step 000.

SR-60 Notes: SR-60 users are invited to share ideas, discoveries, problems, etc. Send your material to Tom Ferguson (421) 3308 N 87th Ave Phoenix, Arizona 85037, who has kindly volunteered to serve as a focal point. If there appears to be sufficient interest, I will publish his consolidations in 52-NOTES from time to time.

Mode Terminology: To clear up some queries, there are really only two operational modes: learn and run (or calculate), and I prefer "RUN" to "calculate" since it is shorter. The machine is functionally in the same mode whether functions are executed manually, SST or under program control.

Membership Address Changes: 248: 4092 Knollwood Orand Blanc MI 48439; 306: 1885 Garfield #9 San Luis Obispo, CA 93401; 347: 2340 Moraine Circle #6 Rancho Cordova, CA 95670.

```
*****    ***         *     *   ****   *****  *****   ***
*       *   *        **    *  *    *    *    *      *   *
*           *        * *   *  *    *    *    *      *
****       *    ****  *  *  *  *    *    *    ***     ***
    *     *          *   * *  *  *    *    *    *         *
    *    *           *    **  *  *    *    *    *      *   *
****    *****        *     *   ****    *    ******  ***
```

Volume 2 Number 3 48/39 March 1977

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

Random Numbers

Gerald Donnely (203) brings up the topic of random numbers, and how best to generate them for particular applications. I should note at the onset that mathematicians are not even in agreement on a definition, let alone how best to generate them! However, for games applications with the SR-52/56 machines perhaps we can agree on the following admittedly fuzzy statements: The "best" random number generator will select numbers one at a time from a given set, producing a sequence whose ordering and distribution characteristics most closely resemble those produced by successive spins of an appropriately marked, unbiased roulette wheel. There are two general mechanization approaches: what I will term open, and closed. Open requires manual intervention via program interrupt; closed is self-contained, and produces sequences which are predetermined by the value of an initializing "seed" number. The open approach comes the closest to the stated concept of best, but imposes a chore on the user; the closed approach produces predictable sequences, but is automatic. The closed approach is used most often, and is generally satisfactory, as most users find it either distastefully ungentlepersonly or too much bother (or both) to determine in advance each predictable sequence. But perhaps the open approach deserves some attention, since it is easily mechanized in-relatively few steps.

If random numbers in, say, the range 1-6 do not need to be hidden from the user, something along the lines of: ***LBL A** 5 = 1 = 4 = 6 = 3 = 2 = GTO A will work quite well: press A, then HLT. The longer the time between pressing A and HLT, the more "random" the selection is likely to be. If the numbers need to be hidden: ****LBL A*** *RCL 05 *EXC 03 *EXC 06 *EXC 01 *EXC 04 *EXC 02 STO 05 *pi sin *ifzro A 0 HLT* might do, and is run with prestored values in Reg 01-06 by repeating: press A and switch from radian to degrees at an arbitrary time. A main program would use the contents of any one of Reg 01-06.

The closed approach poses two problems: how to keep generated numbers within a desired range, and how to get a "good" distribution, all in a minimum number of steps. The severity of the range problem depends on the required range. If the only restriction is that generated numbers fall between 0 and 1, the range problem is trivial. But a requirement, say, to produce a random ordering of the first ten primes might take such a chunk of memory space and/or execution time as to defy practical implementation. But let's look at something in between, As a "working" number, pi is a handy 13-digit, middle-magnitude, positive real whose digit values form a "randomly" distributed string

of 8 of the 10 numerals, and every closed random number generator I've seen for the personal programmables uses it in one way or another. A general approach is to combine pi with a seed by some succession of arithmetic operations, replace the seed with the fractional part of the result, and convert the most significant digit(s) of this result into an integer which may need to be scaled, depending upon range requirements. Each successive generation combines pi with the previously generated seed. Although I've seen square roots and fifth powers applied, a simple multiplication of pi by the seed seems to be quite satisfactory. As an example, let's look at Routine D' of Alan Charbonneau's Yahtzee program (V2N2p5). The requirement is to randomly select one of the numerals 1-6 each time the routine is called. The fractional part of the seed-times-pi product is saved, and then multiplied by the scaling factors 5.5 which guarantees a real roundable to an integer in the 0-5 range. The + 1 offsets the range to 1-6. Although this scaling method works (one of my revisions to Alan's program), Steve Marum (188) points out that there is a bias favoring the numerals 2-6. His revision: 022: RCL 98 X 6 + .5 = ... removes the bias (caution: when making this substitution into Alan's program, either include a neutral spacer, or change all absolute addresses and reposition the unrelocatable code at steps 208-223).

Rather than go on with other requirements and approaches now, I'll wait for input from the membership. Send me your prize random number generators, hard-to-meet requirements, pedagogical arguments, etc and we'll resume discussion in a future issue of 52-NOTES.

A Sum of the Digits Challenge

Jared Weinberger (221) has been experimenting with various approaches to summing the digits of a number. For the SR-52 the sequence: ***LBL A** *(EE INV *fix EE 00 - INV *D.MS INV *D.MS *fix 0 EE SUM 69) INV *ifzro A *EXC 69 *rtn* is Jared's fastest, while: ***LBL B** *X 1 SUM 69 - EE div EE 00 = INV *ifzro B *EXC 69 *rtn* is shorter but slower. Both require a one-time zeroing of Reg 69. Upon call, the displayed number's digits are summed, and the sum returned. For the SR-56, advantage can be taken of the Int function, and the routine starting at 00*: (EE INV *fix EE 00 - *Int SUM 1) *x=t 16 RST *EXC 1 *rtn* is both shorter and faster than either of the SR-52 ones. The routine: *X 1 SUM 1 - EE div EE 00 = *x=t 15 RST *EXC 1 *rtn* is one step shorter, but takes longer to run. To run these SR-56 routines, zero Reg 1 once along with the t register, then RST R/S with the number whose digits are to be summed in the display. Can anyone beat 21 SR-52 steps or 18 SR-56 steps?

PPX-52 Gems and Rejects (52)

Stan Wagon (456) notes that program #390016: Extra Precision Factorials by one Greg Evans "... is interesting and might prove useful to anyone contemplating using double precision in other contexts."

E. S. (Mack) Maloney (246) has a great circle navigation program rejected by TI as a duplicate, but which calculates an initial course from the point of departure (which #949025 reportedly does not). Send Mack a SASE and 25¢ if you would like a copy of his program.

Book Review

Statistics for the SR-52: Keyboard Data, 188 pages $10.00;
Statistics for the SR-52: Card Data, 100 pages $11.00; and
Statistics for the SR-56, 79 pages $9.30; all by C Donaldson Ellis, published 1977 by Wilmardon Publishing Box 461 Storrs, CT 06268.

All three volumes are printed in a pica typewriter font, one side of 8½ x 11 paper, with direct copy of PC-100 printer listings opposite manually generated mnemonics for program listings. Each program begins with a brief description of what it does (assuming user familiarity with statistics usage), along with special requirements and limitations.User instrctions are contained in a step-numbered Procedure section. This is followed by a table of register contents, and the program listing. Applicable formulas and algorithms are noted as being found in one or more of five references, but are not printed in the text; there are no sample problems. All programs are written for use with the PC-100, with notes covering required modifications for SR-52/56 use only.

The SR-52 keyboard manual covers 54 statistics programs, some requiring more than 224 steps. Input data are entered as required from the keyboard. The card manual arranges half of the keyboard programs to allow for data storage and retrieval via mag cards. Data packing and unpacking routines are given, but actual card storage is by program step only (no mention is made of direct storage to/from program registers. Register tables show data assignments to Reg 98, 99, 00, ... 19. I will not comment on the degree to which programs operate satisfactorily or have been optimized, except to note that a 2 X 2/ 3 X 3 matrix inversion program requires 445 steps, and one titled "Four-way ANOVA--Unweighted Means" requires 1736 steps (8 cards).

The SR-56 manual follows along the same lines, covering 40 programs, some of which require more than 100 steps. The longest program is "Mixed ANOVA 1 Within and 1 Between" which requires 242 steps (3 programs to key in).

It would appear that these manuals would be the most useful to statisticians who have access to SR-52/56 machines, but who don't know how (or don't care) to program them. Users are invited to communicate with the author via the publisher.

A Subtle Danger in Using "iferr" as a Flag (52)

Barbara Osofsky (420) notes a potential danger in using an undefined letter function to set an error condition flag in programs where program storage registers are used for data. In my version of her 5X5 program (V2N2p3,4) if any of the first 3 data entries begin with the sequence: 1346 or contain this sequence in the following patterns: XX1346, XXXX1346, or XXXXXX1346, then LBL C becomes defined (see V1N1p4) and the user call to C would cause undesired execution of the code (or data) following the LBL C. While the probability is small that LBL C would be unintentionally defined, the potential is there. As an example of a worst (albeit improbable) case, if the first 2 elements of a 5X5 matrix are: 8.594011346 and -1.234567895D64 respectively, when C is pressed, there is total wipeout! (see V1N2p2). As it turns out, Barbara wat able to accomplish the conditional branch with a real flag in a recent version of her program which I may publish later. In the meantime, users of the V2N2 5X5 program who expect to input matrix elements containing more than 3 digits may wish to make the following

changes: re-write, beginning at step 192: STO 69 + = HLT. (The HLT at step 199 may be either left alone, or deleted). Re-write step 6 of the user instructions to read: 6. Key 9n, press RUN, then E; jth column printed, j=2,3,4,5, see 1 displayed; repeat for n=6,7,8,9. Re-write note 3 to read: 3. At step 6, following the RUN, 5 is flashed.

It should he noted that any time program registers are used for data the possibility of unintentional labeling exists. However, if such registers are chosen at the high end (... 95, 96, 97) duplicate labels pose no problem since the machine always searches for labels starting at step 000.

A Utility Program for Fractured Digits (52)

In modern programming terminology, a utility program is one which aids the programmer in developing other programs. Routine A in V1N5p5 (top) is a short special purpose utility routine that helps the user synthesize look-up tables which become parts of other programs. As noted, once a main program has been written, Routine A is no longer needed, and it can be shelved until such time as it might be used to help in writing another program. A while back I attempted to arrive at the "best" display format for a game program I was working on-by trying out various sequences of fractured digits mixed in with numerical results. I soon realized that it would help to have a fractured digits synthesizer program that could produce specified patterns in the SR-52 display, so they could be seen and judged as they actually appear. In the following program, the user specifies the character he wishes to appear at each of the 14 display positions (within the constraints described in V1N2). 0 produces the numeral 8 (which arbitrarily represents any desired numeral), 2 produces the " symbol, 3 produces a blank, 4 a ' symbol, 5 a °symbol, and 6 a - symbol. Incidently , a 6 for position 11 followed by 3s for positions 12 and 13 will produce a - at position 11 and blanks at positions 12 and 13 (contrary to a statement in V1N2p5). As each position is processed, the number left in the display indicates the next position to be specified.

SR-52 Utility Program: Display Variations Synthesizer Ed

User Instructions:

```
1. Key position 0 code, press A; see 1 displayed
2. Key ith position code, press RUN; see i+1 displayed. Repeat for =1,2,...13
3. Press =, see synthesized display
4. To synthesize a new display, press CLR and go to step 1
```

Program Listing:

```
000:  *LBL B = SUM 98 *LBL C 1 SUM 18 RCL 18 INV EE HLT *rtn *LBL A INV
022:  *stflg 0 INV *stflg 1 - 3 = *ifzro 036 *stflg 0 1.11 STO 98
043:  RCL 92 STO 19 1 STO 18 HLT div 1000 B div 1 EE 4 B div 1 EE 5
069:  B div 1 EE 6 B div 1 EE 7 B div 1 EE 8 B div 1 EE 9 B div 1 EE 10
095:  B div 1 EE 11 B = C - 3 = *ifzro 113 *stflg 1 C div 1 EE 12 B X
121:  10 = STO 18 10 y^x RCL 18 = EE *fix0 *PROD 98 1 EE 88 INV *ifflg 1
147:  151 *1/x INV *ifflg 0 158 +/- *PROD 19 RCL 98 + STO 60 RCL 19
171:  INV EE *fix 0 HLT 0 *ifpos *2' *2' *2' *2' *2' A
```

An AOS Advantage for a Problem in Logic (52)

As most of us know, AOS vs RPN arguments rarely end in conclusive victories for either side. But here is one application where AOS does appear to have a clear advantage. Logician Stan Wagon (456) found that the SR-52's nested parentheses could be made to order hierarchies in certain types of logical statements. In a branch of mathematical logic sometimes referred to as propositional calculus, 2-state variables which assume values of either true or false are operated on by the 5 logical operators: NOT, AND, OR, IF...THEN, and IF AND ONLY IF. All but NOT are "connectives", that is, they link two variables or paired parenthetically grouped variables. A string of variables, logical operators, and parentheses forms a logical statement or sentence. For each possible configuration of variable states, the entire statement is either true or false, and a tabulation of all such configurations is known as a Truth Table. (See Chapter 1 of the Boolean Algebra and Switching Circuits Schaum's Outline if you are new to the subject). Given the elementary truth tables:

			AND	OR	IF... THEN	IF AND ONLY IF
NOT						
A	¬A	A	B	A & B	A □ B	A→B	A↔B
T	F	T	T	T	T	T	T
F	T	T	F	F	T	F	F
		F	T	F	T	T	F
		F	F	F	F	T	T

Stan's Truth Tables program that follows accepts a statement of n variables (0 < n < 10) and up to 27 characters in length, and prints (or displays) the statement's truth or falsity for all 2^n possible ways to assign truth values to the n variables. Each statement as written in LRN mode as the body of Routine E', and must be contained within a parenthesis pair. 4 of the 5 logical operators are assigned to letter-functions as follows: A'=OR, B'=NOT, C'=IF...THEN, D'=IF AND ONLY IF; the "times" operator X=AND. Parentheses should be used extravagantly (when in doubt, use them). The 5 letter functions A, B, ... E correspond to the first 5 variables. For up to 4 more: RCL 06, RCL 07, RCL 08, RCL 09 would be written in Routine E' to represent the additional variables. In order to adopt the output convention that 0=false and 1=true, Stan needed a way to vary the number of displayed digits, and worked out a dynamic-code-change method. A skeleton block of 8 instructions (steps 184-191) held permanently in Reg 93 is modified during program execution to supply the proper value for n in "fix n", then put into Reg 91, which supplies steps 168-175. A logical statement (S) that might be written out as: ((a&b)□(c→¬b))→(a□b) would be written in Routine E' as: ***LBL *E'** (((A X B) *A' (C *C' (*B' B))) *C' (A *A' B)) *rtn.

Each displayed line of the truth table shows the truth or falsity of the statement as an integer 1 or 0. Variable truths or falsities are shown to the right of the decimal. In the above example, the tenths place indicates the truth or falsity of variable a, the hundredths place b, and the thousandths c. The resulting truth tables are displayed as: 1.111, 1.110, 1.101, 1.100, 1.011, 1.010, 0.001, and 0.000, and translate to:

A	B	C	S
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	T
F	F	T	F
F	F	F	F

A printer version of this program would require error test code to terminate execution, in addition to replacement of the HLT at step 179 with a *prt. However, the user could save the extra steps by halting execution manually, and ignoring printout past the ?. Corrections to the preceding page: two references to Routine should have been to Routine E'. An IF...THEN arrow is missing in the written sample statement; the Routine E' code is corrected.

SR-52 Program: Truth Tables — Stan Wagon (456)

User Instructions:

1. Write the statement at label E': GTO *E' LRN (statement) *rtn LRN
2. Key number (n) of variables 0 LT n LT 10, press GTO GTO
3. Press RUN, see ith line of Truth Table. Repeat for i=1,2,...2^n; the 2^nth line is flashed. For new statement, go to step 1.

Program Listing:

```
000:  *LBL A RCL 01 *rtn *LBL B RCL 02 *rtn *LBL C RCL 03 *rtn
018:  *LBL *A' + (STO + 1) X *rtn *LBL *B' 1 + *rtn *LBL *C' + 1 +
038:  (STO - 1) X *rtn *LBL *D' + 1 + *rtn *LBL D RCL 04 *rtn
057:  *LBL E RCL 05 *rtn *LBL GTO *CMs STO 18 +/- EE 70 + RCL 93 =
078:  STO 91 2 yˣ RCL 18 = EE STO 19 *fix 8 1 INV SUM 19 RCL 18 *EXC 00
104:  STO 17 RCL 19 STO 16 *1/x 1 SUM 17 (RCL 16 - 2 yˣ (RCL 00 - 1))
133:  EE *ifpos 143 0 GTO 154 STO 16 RCL 17 +/- INV *log + 1 *IND STO 17
158:  *dsz 114 (*E' div 2 - INV
176:  2 +/- = HLT GTO 091 EE *fix 0 *D.MS *fix 0) X *LBL *E'
```

TI Notes

SR-52 and SR-52A Trouble Shooting Guide: reported by Marshall Williams (277) and later confirmed by TI as available to the public. Send $11.00 plus local sales tax to TI Box 53 Lubbock, TX 79408. This manual reportedly contains schematics and I/O interface information.

PC-100 Modifications: to upgrade to PC-100A reported by P S Cox (9) and later confirmed by TI as a possibility. If implemented, TI would modify PC-100s for an as yet to be determined fee to accomodate "other" machines, but such modifications would not include the battery charger feature of the PC-100A.

Routines

An Efficient Input Routine (52): Ed Parsons (65) has found a way to save 2 steps for a routine that sequentially stores input data and displays a data counter. The usual approach is along the lines of: ***LBL A** *IND STO 69 1 SUM 69 RCL 69 HLT. Ed replaces the RCL 69 with +, which works just as well, provided processing begins with CLR or =.

A Short Integer Test: Stan Wagon (456) suggests the sequence: ... - *D.MS = *ifzro ... as a quick test to determine whether or not a number is an integer. However, one limitation is that the number must be less than 13 digits for older machines (see V2N2p6). A comparable SR-56 sequence would be: ... - *INT = *x=t, with a zeroed t register.

Volume 2 Number 4 48/39 April 1977

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

Editorial: A Perspective On Machine Comparisons and Loyalties

Invitation: To Engage In Friendly Competition

During the two-years that the HP-65 reigned supreme as the first and only pocket (personal) programmable calculator (PPC), there was no question of loyalty to a specific machine or manufacturer, or of measuring machine capability against a competitor's. If you wanted a PPC, the HP-65 was it. Back in late 1973 those of us who had some inkling of what a PPC might do could hardly wait for the first one to become available, and stood in line for months for the privilege of paying $800 for an HP-65. We were primarily motivated by two expectations: 1) Having exclusive possession of a considerably more powerful computational tool than ever before, and 2) Having the opportunity to win the recognition and acclaim of our peers which would follow discovery and invention.

I submit that these same two expectations continue to dominate our thinking as we attempt to choose frqm a growing proliferation of new PPCs. Most of us who already had HP-65s and who still decided to buy SR-52s over a year ago, were unaware of user access to as many as 60 storage registers, yet saw the power of, and could imagine the potential for inventiveness offered by the IND function. Then, having both machines made us face up to the matter of machine/manufacturer loyalty. I had a strong bias in favor of HP machines, having worked extensively with the 3 HP PPCs: 65, 55, and 25. And although I still prefer the HP machines for manual use, when the HP-67 appeared, it didn't seem to have enough new programming features to lure me into buying it, especially since it lacks some important programming capabilities which the SR-52 has. Since each manufacturer is motivated to design new machines based on what it thinks the market wants and that it can sell for a profit, it would be surprising if one manufacturer consistently produced machines with the greatest discovery/invention potential. So I see no point in restricting one's patronage to any one company or product line in this game.

Unfortunately, excessive loyalty toward one machine makes it difficult if not impossible to compare competing macnines impartially, or even correctly. And I suspect that all else being equal, uni-loyalty becomes the strongest toward the most expensive of competing machines; the rationale goes: it just has to be better since I paid more for it.

I bring up this matter of machine loyalty because I think it helps to explain some of the poor comparisons that abound in print, and to suggest that there may be a more productive and stimulating way to compare rival PPCs than by extoling (or quashing) virtues: Write and demonstrate tough programs on one machine and challenge users of a rival machine to match or better them. To be sure, there will always be some disagreement over what is "best", but there are probably enough solid judging criteria such as: required number of keystrokes, precision of results, technical-validity, execution time, I/O ease, max number of inputs, handling of worst-case conditions, etc to make the outcome of programming contests conclusive in most cases.

I have discussed the idea of friendly competition with Richard Nelson (Editor and publisher of **65-Notes**), and we have agreed to give it a try. Machines would generally be matched-as follows: SR-52 vs HP-65 or HP-67; SR-52/PC-100 vs HP-97; SR-56 (without printer) vs HP-25; although other combinations might be appropriate. Richard and I will do the judging, calling upon tie-breaking expertise as required. Users of TI machines should send candidate challenger programs to me; HP to Richard. We will decide jointly which challenger programs to accept, and announce them in our respective newsletters along with suspense dates for response. Letting better challenger programs supercede original ones may be allowed, but changes would be subject to mutual agreement. For the machines with card readers, candidate programs must fit on one card, although manual code modifications will be allowed and will count in the total of required user keystrokes. Data may be read in via additional cards. For the other machines, there will be no upper limit on program size; overflow will automatically incur the penalty of additional user keystrokes. All programs will be subjected to mutually agreed upon test cases. Additional rules, requirements and constraints will be announced as they suggest themselves and are mutually accepted. To get the ball rolling, to Richard I suggested Barbara Osofsky's SR-52/PC-100 5 X 5 Determinant and Inverse program (V2N2) as an HP-97 challenger, and he has accepted. The particular version of Barbara's program has yet to be decided.

The restricted and general case versions of "Corner The Lady" (see A Game Challenge below) will likely be good subjects for future challenges, but I prefer to let them get used and improved upon for a while first. In the meantime, send me your candidate challengers: programs you've written for SR-52, SR-52/PC-100, or SR-56 that you think would be difficult to program on the HP-67, HP-97, or HP-25, respectively; and get ready for challenges from the other side!

More on Printer Graphics (52)

As several of us noticed, the 17 March 77 issue of **Electronics** has an article describing various approaches to and uses of SR-52 printer graphics (see V1N3p2), some of which appear to be worth trying. However, the listed program is not very well optimized, and could easily be improved for faster execution, and input ease. Our thanks go to the author for the 52-NOTES V1N3 program credit.

Membership Renewals

Although I will continue my policy of not issuing individual reminders, I'll note that many memberships become due for renewal in May. If you are in doubt as to your status, send me a SASE.

A Game Challenge

Martin Gardner appeared to be throwing down the gauntlet to SR-52 users when he stated in his Mathematical Games column in the March 1977 **Scientific American** that "...It is easy to program an HP-97 printing calculator or the HP-67 pocket calculator to play a perfect game" without mentioning possible SR-52 mechanization. The game is one related to the NIM family which Gardner calls "Corner The Lady". As it turns out, the HP Games PAC program he refers to (called Queen Board by HP) handles only the restricted case of an 8 x 8 standard chess board for the playing field, with each of the 64 squares numbered in a way that facilitates processing, but which is rather cumbersome for the user and decidedly unconventional for rectangular notation. In any case, the 8 x 8 game should indeed by easy to mechanize on an HP-67, since a modest effort produced an SR-56 program (below) that is more user oriented, and still leaves program steps and data registers to spare. The real programming challenge lies in mechanizing the algorithm that applies to any size board, to produce a program that will fit within 224 SR-52 steps. I had to try several approaches before arriving at the SR-52 program that follows, which seems to work, but is slow. Since finding enough program memory was the biggest problem, and since an integer/fraction routine is required, and also since there are data registers to spare, I suspect that this may be one case where the HP-67 has a decided advantage over the SR-52. However, there may be alternate viable approaches that would make better use of SR-52 features. I have purposely left the code in relocatable form, and invite the membership to improve on it before someone writes a better HP-67 program!

Gardner's article delves into some interesting facets of Fibonacci numbers, and I expect many of you will find the time spent perusing it to be worth while. For my approach, the ten-digit display size limits the maximum playing field to about 100 X 100.

This is reduced further to about 60 X 60 to gain an extra program step. I say "about" in each case since there is a position dependency caused by the conversion of base ten numbers into Fibonacci Notation in non-canonical form, as max board size is approached. The game is played on a standard or enlarged chess board (the squares need not be of alternate color) with one queen as the only playing piece. Starting at any square along two adjacent edges of the board, the queen is moved alternately by each of two players toward the corner opposite the corner formed by the starting field edges. Moves are along files or diagonals as in chess, but the queen may not move backwards or "sideways" (such that no progress is made toward the corner goal). The player that moves the queen into the required corner wins. For the two programs that follow, the playing field is addressed by standard x,y notation, such that for the 8 x 8 case, the starting field is the set of squares: k,7 and 7,k where k=0, 1, ...7, and the winning square is 0,0. Advantage is taken of symmetry, and actual x and y values are translated into playable numbers via a and b as defined. The SR-56 game is definitely more "playable", and should be easy to convert to SR-52ese if you are more interested in playing the game than in mechanizing the generalized algorithm.

SR-52 Program: Corner The Lady Game (Fibonacci Notation) Ed

User Instructions:

1. Key a < b < 55, press A, see 0 after about 20 seconds
2. Key b, press RUN; if display flashes, see b'; (a'=a)* else, see a'; press RUN, see b'

*if b'=b, machine's move is a, b-1
For x=a, y=b, x'=a', y'=b'; for x=b and y=a, x'=b' and y'=a'
a,b = player's move; a',b' = machine's move as translated for play

Program Listing:

```
000:  *LBL D 1 + 5 *√x = div 2 = y^x RCL 00 div 5 *√x = *fix 0 *D.MS *rtn
022:  *LBL E + 1 = STO 19 CLR 9 STO 00 *LBL *2' D *dsz *1' RCL 69 *rtn
044:  *LBL *1' - RCL 19 = *ifpos *2' +/- STO 19 RCL 00 INV *log *D.MS
063:  SUM 69 GTO *2' *LBL C RCL 18 div 10 -  *fix 0 *D.MS STO 18 = *rtn
085:  *LBL B ST0 18 CLR 1 STO 00 *LBL *5' C *ifzro *6' D SUM 69 *LBL *6'
106:  1 SUM 00 RCL 00 - 10 = INV *ifzro *5' RCL 69 *rtn *LBL A STO 98 E
130:  STO 17 STO 18 0 HLT STO 99 E STO 16 1 STO 15 *LBL *3' 1 +/-
153:  *PROD 15 C *ifzro *3' RCL 15 INV *log *fix 1 EE *PROD 17 RCL 15
173:  INV *ifpos *9' RCL 16 - RCL 17 = *ifpos *9' RCL 99 - RCL 98 - 1 = E
197:  + 1 = X 10 = STO 99 B HLT RCL 99 X 10 = B HLT *LBL *9' RCL 17 B
```

SR-56 Program: Corner The Lady Game (Standard Chessboard) Ed

User Instructions:

1. Initialize: press RST, R/S
2. Key a.b 8 GT a GT b, press R/S see new position: a'.b' after machine's move. Flashed 0.0 indicates machine's win, for new player's move, go to step 2. For new game, go to step 1

Program Listing:

```
00: R/S *Subr 72 *x=t 63 STO 1 - *Int STO O *CP = STO 2 *x=t 67 X 10 =
24: x:t RCL 0 *x=t 67 INV *dsz 48 RCL 0 + RCL 2 = *Subr 72 INV *x=t 30
47: RST RCL 1 - 1.1 = *Subr 72 INV *x=t 50 RST - 1 = RST 0 *fix 1 + =
72: x:t 1.2 *x=t 95 2.1 *x=t 95 5.3 *x=t 95 7.4 x:t *rtn
```

Book Review: SR-52 and SR-52A Troubleshooting Guide by Jim Johnson, Kevin Hines, Donna Davis, Don Young, published by TI (see V2N3p6); 32 pages plus two fold-out schematics.

This booklet is of mimeograph-quality print on 8½ x 11 paper, one side only. Most diagrams are handsketched; some detail is difficult to read. The stated purpose "... is to describe the operational theory of the SR-52 and SR-52A and to give guidance in its repair". Theory of operation covers: power supply, clock, the MOS chip set, keyboard, display interface, motor control, card sensor input, mag read/write mechanism, and a reference circuit. Illustrations include the keyboard matrix, display wiring, motor drive circuit, card sensor circuit, reference circuit, SR-52A Block Diagram, 3 pages of waveforms, an unlabeled page of step functions associated with the display digit select signals, four pages of component layouts, 7 pages of parts lists, and 2 fold-out electronic schematic diagrams. Since I am not a EE, I will defer technical review to whomever feels qualified and would like to volunteer. Members able to put any or all of the info in this manual to good use are invited to share their successes via 52-NOTES.

Correction

Step 193 of the SR-52 program on the previous page should be - not =. [RAHP: Corrected]

Pseudo 83: Part I: Accessing the Instruction Address Register (IAR)

Of all the pseudos, 83 stands out as the most fascinating and enigmatic. Orla Damkjer (393) has examined p83 in considerable detail and reports a wide variety of results from its execution, with interrelated dependencies upon such things as the number of digits in the display following the decimal point, display notation/format, angular mode, soft or hard display, whether or not p83 execution has been prefixed by CLR or rset, and even at what step the p83 has been executed. This last dependency was also discovered by Jim Rosenberg (516) who notes that it provides a means of access (albeit somewhat gross) to the IAR. This will be the first topic covered in a series on p83. Members are invited to explore this and other facets of p83 behavior that might lead to new programming capabilities, and to share their significant results. A word of caution: p83 behavior is so dependent upon so many pending conditions or machine states that it is easy to make false assumptions about experimental control conditions.

An IAR is a computer register that tells the machine what instruction to execute next by containing the address of the next instruction. It is incremented during sequential execution, and fed new data for branches. The PPCs all have IARs, but their contents are not accessible to executing programs. When computers are operated in a real-time environment, requirements for program execution to respond immediately to unpredictable external events make it necessary to be able to determine where in a main program, execution stops when interruption occurs. General purpose computers are designed with one or more so-called interrupts which cause program execution to branch to one or more predetermined step locations following completion of the instruction being executed at the time an interrupt occurs. Special interrupt processing routines are written and positioned starting at this (these) predetermined locations to save important information that is vital to continued main-program execution. Included in this saved information is the contents of the IAR, so that after the requirements posed by the interrupt have been met, proper transfer back to the main program can be made. While it may be hard to find practical uses for SR-52 IAR access, mechanization is fairly challenging, and illustrates an important general purpose computer function.

When p83 is executed with 4 digits following the decimal point in the display, a zero divide error state without flashing display appears to result (see V1N1p2) and information is stored in Reg 60 that identifies which program storage register contains the just-executed p83 step. Further, if a program-executed subroutine call precedes the p83 execution, with no intervening branches or rsets, Reg 60 also contains information identifying which program storage register holds the first executable instruction of the subroutine. If we examine the contents of Reg 60 as it appears in the display following a subroutine call - p83 sequence, the following characteristics are observed: the number is either zero, or it is in scientific notation with negative mantissa and negative decapower, in the form: -m. -dd or -m.m -dd . The mantissa identifies the adress (ss) of

the program storage register containing the first executable instruction of the called subroutine by the relationships: ss=m+70 or ss=10(m.m)+70. The decapower identifies the address (pp) of the program storage register containing the just-executed p83 by the relationships: pp=dd+48 if the mantissa is in the form -m., and pp=dd+49 if in the form -m.m . However, if the first executable instruction of the called subroutine is in Reg 70, the mantissa is zero which causes the display to be zero, and there is no information concerning the location of the just-executed p83. Also note that when m is either 1 or 10, it is displayed as 1. As an example, program the sequence: ***LBL A** E p83 HLT starting at step 072 (which puts the p83 in Reg 79), and the sequence: ***LBL E** 1.2345 *rtn starting at step 215 (which puts the 1 in Reg 97). Press A, RCL 60, and see -2.7D-30, which by the above relationships confirms the locations of the 1 and the p83. To make use of such Reg 60 information during program execution, use the Weinberger exponent extractor (V1N4p6) with rounding, and extract the mantissa with: EE 00. Because of possible ambiguities, a Reg 60 processor routine needs to be written such that the pre-p83 called subroutine is known to be either in the Reg 70-79 block or the Reg 80-97 block.

While all this appears to work under the stated conditions, there may be subtle machine states that go unnoticed which could cause trouble. I've found that if you make the subroutine call manually, or eliminate it, the Reg 60 information will be in a different form, and there are likely to be many more special pre-conditions that will affect results. It might be interesting to write a program that is D/R switch interruptable at many points such that an interrupt processor routine determines within a block of 8 steps where an interrupt occurs, and where the last called subroutine is located.

SR-56 Program Exchange

Dave Johnston's 1 April 1977 catalog lists 21 math, 4 statistics, 23 physics, 7 games, and 6 misc programs for a total of 61. Members who have found particular SR-56 programs to be especially helpful, well-written, etc are invited to share this information via 52-NOTES. Send me the name, Dave's catalog number, and a few lines telling what you like about them.

TI Invitation to Club Members

A letter from Robb Wilmot, Manager of TI's Professional Calculators Division, suggests that the mid-June National Computer Conference to be held in Dallas this year might be a good occasion for us users to hold *"... informal discussions with some of our (TI's) design and applications people.*

"You are aware that, due to our designers work pressures, we do not normally encourage their dialogue with users groups, but the NCC Show being in Dallas, I feel, offers a unique opportunity for a 'once in a while' dialogue."

I have responded that our interest in participating in such dialogue would be determined in large part by the extent to which TI would expect to discuss topics of substance. I will pass along further TI response via future 52-NOTES.

Volume 2 Number 5 48/39 May 1977

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

Quasi-Alphanumerics: A Comparison Of SR-52 And HP-67 Capabilities

A few months ago an exciting HP-67 discovery (consequences of which are still unfolding) opened up the possibility of displaying quasi-alphanumeric messages on that machine... a capability with similarities to SR-52 fractured digits and which got me to wondering about using a mixture of fractured digits and numerals for SR-52 messages. I don't think the value of displayed messages needs to be defended, but as an insight into what matters most with limited capabilities, it is instructive to examine what each machine can do. For purposes of this discussion, display positions for both machines are numbered from left to right as 0-13. An HP-67 user can store and recall special "numbers" to and from data registers which when displayed can show the ten digits, facsimiles of the letters: r, C, o, d, and E, the minus sign, and blank, in accordance with what appear to be the following rules: Position 0 can only be either blank or the minus sign, each of the positions 1 through 10 can be any character except the minus sign, position 11 can only be blank, and positions 12 and 13 can be any character except the minus sign, but their combined value determines the position of the decimal point (which cannot be suppressed). Once a complete display line has been synthesized and successfully stored in a data register, it cannot be merged with changing data; i.e. selected positions cannot be masked. However, there is a way to modify pure data to be flickerdisplayed with minus signs, blanks, and decimal points. Messages can be displayed effectively using the two built-in pause functions, a software-generated flicker pause, and the data-entry-sensing capability that flag 3 provides: automatic continued program execution when a number is entered from the keyboard during a pause-displayed prompting message. It is apparently by design that the 5 fractured digits look like letters, since they are all used in the 2 words *Error* and *Crd* generated by firmware and displayed as user cues.

So far as I know, in addition to the blank, the SR-52 can produce only 4 fractured digits, only one of which (the square o) is very letter-like. But before we dismiss the SR-52 message-display potential as being too limited, let's see what can be done. I have experimented using the following character assignments: A=°, B=8, D=0, E=3, F=', G=9, H=", I=1, J=', L=1, N=", O=0, P=6, R=7, S=5, T=-, U=", Y=4, Z=2, and find that when these symbols are grouped in the display to form familiar words, intended meaning is generally conveyed. But it's probably best not to hold to fixed character assignments, as specific juxtapositions may suggest variations. For example, by the

above assignments, the word GOOD would be spelled 9000. But 9℃0 is probably more recognizable, even though it might, in certain contexts, be interpreted as GOAD.

The message synthesizer program below may be used to see what various trials look like, and it produces the 2 numbers required to produce the message by a running program using the sequence: RCL 98 + STO 60 RCL 99 HLT (with = pressed manually). After a desired message has been synthesized, the contents of Reg 98 and 99 can be transferred to 2 program registers (unused by a user program) for permanent storage. 14 characters are entered as single digits followed by A for the first one, then either E (if to appear normally) or C (if to appear fractured). The fracture codes are per V1N2p5. Positions 0 and 11 may only be either blank or -, position 10 is only zero for the SR-52A, else must be one of the ten digits (0-9), and zeros in positions 12 or 13 are displayed as blanks. For example, try: 6A, 7E, 4E, 3C, 5C, 9E, 5C, 1E, 2C, 3C, 0E, 6C, 1E, 5E; then after you press =, see: TRY AGAIN OTIS. Or (SR-52 only) 3A, 9E, 3E, 6C, 3C, 5C, 2C, 3C, 3C, 5E, 7E, 6C, 5E, 2E; Press = and see GET AN SR-52.

One can, ofcourse, mix messages with changing data. During the synthesis process, substitute zeros for the positions to be dynamically filled, and design the user program to put the desired data into these positions via careful register summing for positions 1-10, and and multiplication for positions 12 and 13. Keep a copy of the original "template" (with the pre-positioned zeros) intact, as a new template will be needed for each data change.

Out of all this, it is evident that the limitations of each machine are quite different. The HP-67 fractured digits are more letter-like, but there are fewer "natural" multiple assignments (such as " standing for H, N, or U for the SR-52); the SR-52 can mask selected positions, but display isn't as automatic or fancy (pausable or flickerable); each HP-67 line takes considerably more effort to produce, but each can be coded into the contents of a single data register, recorded, and produced (displayed) with a single instruction (RCL). Neither the HP-97 nor the SR-52/PC-100 can print fractured digits.

Much of the incentive for trying to create messages on either machine is due to the challenge: it is mildly difficult, and often considerably frustrating. But, of course, as soon as a PPC comes along with real alphanumerics, nobody is going to bother, any more than we would spend time now developing efficient manual keystroke sequences for the non-programmable scientifics. But it should be fun for awhile! So send me your best messages and/or applications, and perhaps quasi-alphanumerics will become a lively part of the TI/HP users challenge.

SR-52 Program: SR-52 Message Synthesizer Ed

User Instructions:

1. Key position 0 code (3 or 6), press A, see 1.
2. Key position i code, press C for fracture, E for normal, see i + 1; repeat for i=1,2, ...13.
3. Press =, see message.
4. To transfer message code to another program:
   A. Write other program into memory
   B. In RUN mode: Key RCL 98 STO ab RCL 99 STO cd, where ab is the address of program register to hold operator number and cd the operand register.

SR-52 Message Synthesizer Program Listing

```
000:  - 6 = *ifzro 009 *stflg 0 CLR 9.00 STO 98 STO 99 .01 STO 69 9 STO 00
030:  11 STO 68 B *ifflg 1 053 *E' SUM 99 *dsz 035 GTO 061 *E' SUM 98
057:  *dsz 035 B *E' SUM 99 B - 6 = *ifzro 076 *stflg 2 B *ifflg 1 096
082:  - 1 = X 10 = SUM 68 GTO 103 EE +/- 12 SUM 98 B *ifflg 1 119 - 1 =
112:  SUM 68 GTO 131 X 10 = INV *log *fix 0 EE *PROD 98 RCL 68 INV *log
136:  *fix 0 EE *ifflg 2 146 EE +/- *PROD 99 *ifflg 0 159 1 +/- *PROD 99
159:  RCL 98 + STO 60 RCL 99 HLT *LBL *E' X RCL 69 = *rtn *LBL E INV
181:  *stflg 1 *rtn *LBL C *stflg 1 *rtn *LBL B .1 *PROD 69 1 SUM 67
200:  RCL 67 INV EE HLT *LBL A = *rset
```

SR-52A/PC-100A

Peter Stark (321) notes that while the SR-52A has been alluded to in 52-NOTES, it has not been properly identified. It is the "newer" SR-52 I refer to on V2N2p6 under "Display Arithmetic Modification" and so far as I know, such machinbs are not physically marked as such ("SR-52A" is not printed or stamped on the machine). Bob Edelen (100) notes that Alan Charbonneau's Yahtzee program (V2N2p5) will not work with an SR-52A, and suggests the following changes for use with the newer machines: replace the 5 with 6 at step 98; replace the sixes at steps 210-214 with ifflgs. All five dice will show, followed by an irrepressible zero. While the TI trouble shooting manual (V2N4p4) suggests a number of hardware differences between the SR-52 and SR-52A machines, the only difference I am aware of that is noticeable to the user is the 13th digit truncation.

Bob Edelen has made some electric power measurements and reports the following: the PC-100A charges the battery PAC at a constant 120ma even when the power switch is off. His SR-52A with adaptor charges a full PAC at 100 to 110ma with SR-52 off; on: 50ma. A dead PAC draws 120ma with SR-52A off; on: 60ma. SR-52A turned on with 0 displayed draws 160ma, with full display 260ma, which discharges a full PAC 45ma during charge. SR-52A execution (with the 2 minus signs displayed) draws 145ma. Since continuous charging is not good for batteries, Bob has modified his PC-100A to allow the SR-52A to discharge the PAC while connected to the printer. Members interested in details are invited to send Bob a SASE (note his address change elsewhere in this issue).

TI Notes

Machine Exchange at TI Centers: Although individual retailers may be changing their exchange policies, it is still TI policy (see V2N1p2) for its regional centers to continue to exchange good machines (of any model vintage) for bad, that are still in waranty.

Adding Chips (52): TI is now making available to the public the TMC 0599 chip via Mr S Riggs, Box 53, Lubbock, TX 79408 for $17.77 each. Reportedly, 2 of these will provide the missing SR-52 registers (20-59) if properly installed. Michael Rak (502) has obtained details from a TI source, and if you are experienced at assembling modern electronics components and are willing to risk machine damage, send 25¢ and a SASE to Mike at 1823 El Cerrito Pl #F Los Angeles, CA 90068 for a copy of his info. Neither Mike, nor TI nor the SR-52 Users Club assumes any responsibility whatsoever for any consequences which might arise from chip ad-on mods. TI does not now, and does not plan to perform this mod for the general public.

Tips

Mag Card Marking: Noting that the first issue of the PPX-52 newsletter mentions use of a "Sharpie" pen, Mack Maloney (246) has found that the Sanford Sharpie #49 is permanent, and that a Vis-a-Vis 69 is smearless with markings that can be removed with a moistened fingertip or cloth, and which seem to work in place of the write-protect mag card tabs.

Another One-Step Error Producer: Jared Weinberger (221) suggests repeating an operator to create an error condition. For example, the sequence: RCL 99 + + RCL 98 = performs the desired addition as well as creating an error condition.

More on Trig Function Anomalies: Karl Hoppe (507) notes that attempts to take the sin, cos, or tan of a number whose absolute value is between zero and 3.6 D-97 in degree mode (or 6.283185307180 D-99 in radian mode) result in an error condition. This may be due to the creation of underflowed intermediate results by the trig firmware.

IND Prefix to a Subroutine Call (52): Several members have noted that placing an IND before a subroutine call has not yet been mentioned explicitly in 52-NOTES. An IND prefix maintains viability through a subroutine call, just as an INV does (V1N1p2). Further, the sequences: INV *IND or *IND INV are viable together. For example, the routine: ***LBL A** SUM 99 *rtn prefaced with INV *IND will cause the contents of the register pointed to by Reg 99 to be decremented.

Battery Charging PAC: Noting that TI does not supply a battery charging PAC for any of its PPCs, Bob Edelen (100) claims that the HP PAC will work with connector mods, and a rubber band to hold things together.

More on the 0 Divide Error Condition (52): Howard Cook (556) reports that an underflow resulting from dividing a small number by a number ≥ 10 produces the 0 divide error condition (see V1N1p2). However, product by a number ≤ .1 does not do the same thing.

Fast Relative Addressing (52): John DeHaven (548) notes that the machine will branch to a label located within the first five or so steps faster than it will with an absolute branch.

Routines

Odd/Even Determiner (52): Jim Rosenberg (516) has devised a clever way to determine whether a positive integer is odd or even: ... STO 00 50 *PROD 00 *IND RCL 00 *ifzro GTO ..., which branches to GTO on odd, continues on even. Here is another way a non-register (50) can be put to good use (see V2N2p6).

Efficient Temperature Units Conversions (52): John DeHaven (548) has an algorithm for conversions between Centigrade and Fahrenheit which combines the 2-way calculations efficiently and which can be mechanized by: ****LBL A*** **E'* ****LBL B*** *+ 40 = X 1.8 *iferr* ****LBL *LBL *1/x*** *- 40 = CE HLT.* with temp in degrees F in the display, press A for conversion to C; B for C to F. (E' must not be defined).

Membership Address Changes

100: 1632 Yosemite Denver CO 80220; 128: 356 Dart Dr Hanover PA 17331; 252: 2111 Bidwell Rd #C1 Muscatine, IA 52761; 353: 2725 Monte Mar Terrace Los Angeles, CA 90064.

Friendly Competition

The version of Barbara Osofsky's 5 X 5 Matrix program currently the challenger for HP-67 users lists below (see V2N4p2), and needs no further comment (see V2N2p3), except that I take credit only for using Rick Wenger's error-flag approach; credit for eliminating more keystrokes and the potential wipe-out danger (V2N3p3) goes to Barbara.

Another challenger candidate is Karl Hoppe's (507) "Square of 70 Digit Factor" SR-52 program, which follows. While there might possibly be a sufficiency of data registers, ten-digit arithmetic will be a hurdle for HP-67 mechanization. One construct which Karl uses identifies an EE viability of which I was unaware: Decapower display softening by EE carries through a subroutine return into main-program continued execution. For example, the call to D at step 112 returns 1D21 in the display register, which then gets modified to 1D16 by the 6 at step 113.

SR-52/PC-100 Program: 5X5 Determinant and Inverse B.Osofsky/Ed

User Instructions:

1. Initialize: Key 95, press C
2. Key e_i, press E; repeat for i=1,2,...25 with column-wise catenation; see 1 displayed.
3. Get determinant: with 1 in display, press D; first col of adjoint printed; determinant displayed.
4. (optional) Print Determinant. On printer: press ADV, PRINT, ADV
5. Get Inverse: With determinant in display, press D; first column printed, 1 displayed.
6. Get ith column: Key i, press RUN, repeat for i=2,3,4,5; ith column printed, 1 displayed. For new problem, go to step 1.

Program Listing:

```
000:  *LBL *B' STO 68 (*IND RCL 66 X *IND RCL 68 - 1 SUM 66 +/- SUM 68
024:  *IND RCL 66 X *IND RCL 68) X *LBL *E' 1 SUM 66 *rtn *LBL C STO 66
047:  *rtn *LBL *C' (8 C 4 *B' *E' 17 *B' +/- + 19 *B' C 7 *B' - 14 *B'
071:  *E' 17 *B' + 19 *B' *E' 2 *B' + 9 *B' *E' 2 *B' - 4 *B' 6 C 12
095:  *B') div RCL 67 +/- STO 67 X *prt RCL 95 + *rtn *LBL *D' 10 +/- A
118:  *LBL B 95 C *A' *A' *A' *A' *LBL *A' *IND RCL 66 *IND EXC 69
137:  *LBL E *IND STO 66 5 *iferr 149 1 SUM 66 *LBL A SUM 69 *rtn
158:  *LBL D STO 67 CE 0 STO 69 *C' B *C' B *C' B *C' B *C' 5 A
180:  *D' *D' *D' *D' *pap 0 = HLT STO 69 94 A *x! B GTO 164
```

SR-52 Program: 70 Digit Square Karl Hoppe (507)

User Instructions:

1. Initialize: Key 99, press B
2. Input up to a 70-digit positive integer (no decapower); Key blocks of ten digits, each followed by RUN; begin with MSD; see prior block of ten digits (from earlier problem).
3. Get square: Press A; after about 200 seconds, see the first ten MSDs of the square.
4. Get remaining digits: Key 6 B; then key RUN for successive blocks of ten digits. For new number, go to step 1.

70 Digit Square Program Listing:

```
000:  *IND *LBL E RCL 98 *rtn *IND *E' 1 +/- C *rset *LBL D - (STO +
019:  *LBL *D' 1 EE 21 *rtn *LBL C +/- SUM 98 *LBL *C' SUM 68 *rtn
038:  *LBL *B' RCL 68 - 6) *rtn *LBL *A' 18 STO 68 19 *LBL *E' STO 98
061:  *rtn *LBL A 105 STO 69 *A' 0 *IND *E' *IND STO 68 2 C *B' *ifpos
082:  071 9 *C' 6 *C' 105 *E' *IND RCL 69 D 6 - *D' 6) STO 67 = STO 66
110:  *IND E D 6 - *D' 6) STO 64 STO 65 = STO 63 X RCL 67 *PROD 64 +
136:  RCL 65 X RCL 66 *PROD 63) D - *D') SUM 64 + RCL 63) *IND *C' 1 C
163:  RCL 64 div *D' 0) *IND *C' E - 99) *ifpos 110 SUM 69 *B' INV *ifzro
187:  087 *A' *IND E + *D' - *D') *IND INV SUM 98 div *D' 0) *IND *C' 1
210:  C *B' *ifpos 191 CLR 6 *LBL B *E' *rset
```

PPX-52 Gems

#300032A: 4 X 4 Matrix Multiplication, by Barbara Osofsky (420), facilitates chain multiplications by replacing each successive multiplicand by the next product. The PPX program analyst's note (2) is in error, likely to be misleading, and should read: "Insertions and deletions in the program will change the first 10 entries of matrix 1". As Barbara notes, warnings of this type should be made for any program that uses program registers for data, since data stored in program registers are equivalent to unrelocatable instruction code.

#910050A: Two to Nine Pile NIM, by R L Floyd (82) effectively mechanizes the modulo two addition algorithm that handles up to nine piles, each of which can be up to 1D1O in size. Richard was the only one to respond to my invitation (V1N3p3), but noted at the time that he was submitting his program to the PPX-52 library.

Book Review: Computers At Large, by C J Sippl (239) and R Bullen, Bobbs-Merrill, 1976, 222 pages.

Here is a readable discussion for the layman that covers a broad spectrum of man-computer relationships. Emphasis is on the sociological, psychological and philosophical impact computers have made and will make on man and society. The text is liberally sprinkled with quotes which the reader can pursue in greater depth via a substantial bibliography. Discussion of the varied uses to which computers have been put, and the broad range of consequences to humans provides good food for thought.

Meeting the Sum-of-the-Digits Challenge (V2N3p2)

Howard Cook (556) beats Jared's shortest by one with: ***LBL C** EE STO EE 00 - 1 SUM 69 = INV *ifzro C *EXC 69 *rtn, and notes that Jared's routine A and his routine C effectively sum the rounded 10-digit display, and that Jared's routine B sums up to 12 of the most significant digits in the display register. Howard goes on with a new challenge: beat his 56 step routine D which sums all 13 digits:

```
*LBL A (STO - .5) EE INV EE *rtn *LBL D STO 01 *fix 0 *log A INV *log A *1/x
*PROD 01 *LBL RCL RCL 01 *ifzro *EXC A SUM 69 INV SUM 01 10 *PROD 01 GTO RCL
*LBL *EXC *EXC 69 HLT,
```

which he says is clumsy, but which has some clever constructs.

Subject/Author Cross Reference

Bob Edelen (100) has kindly volunteered to work up an index covering the first 12 issues of 52-NOTES, and will cut mimeo stencils under computer control, in much the same way Mike Brown did with the alphabetized membership list. I'll be asking Mike for a follow-on soon, that will cover only currently active members.

Volume 2 Number 6 48/39 June 1977

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

The New TI PPCs

The speculation is over (for awhile) following TI's announcement the last week in May of the New TI Programmable 57, 58, and 59 calculators. The 57 looks like even tougher HP-25 competition than the SR-56, with 10 labels and an $80 list price; the 58 and 59 are something else, packing some fascinating new features that it is hard to believe can be squeezed into a slightly smaller package than the 52, and at list prices of $125 and $300 respectively. Both use the same instruction set, accept 5000-step plug-in read-only-memory (ROM) program modules, and plug into the PC-100A printer; the 58 has half the 59's memory, and no card reader. The discussion that follows applies to the 59, but is generally applicable to the 58 except for the differences, just noted.

120 general purpose memory registers are partitionable between data and program storage use: 1 data register is equivalent to 8 program steps from a 0/960 data/program mix to 100/160 in partitionable increments of 10 registers (11 possible configurations). Once a partition has been established, boundaries cannot be crossed (as with the 52). Partitioning/repartitioning can be done from the keyboard or under program control, except when a protected (in the proprietary sense) card has been read, in which case the user cannot get into LRN mode, single step, or recall as data, or copy on to another card the program memory contents.

The 5000-step ROM modules (which TI calls CROM for Constant ROM) are accessed by program number, and run by subroutine name for execution through the keyboard or under user-program control. The Master Library (which comes with each machine) holds 25 programs from a 52-step D.MS add/subtract/scaling routine to an 898-step matrix determinant/inverse/simultaneous equations leviathan that will handle up to a 9 X 9 matrix, or a system of 8 equations. Once a CROM program has been accessed from the keyboard, subsequent pressing of user-defined keys (A-E') refer to the CROM program subroutines, but a CROM program must be accessed each time a subroutine is to be called under user-program control. CROM programs may be down-loaded into user-program memory for inspection and/or editing, and run as user programs, but not the reverse. Configuring new CROMS is an expensive manufacturing process, and is not a practical user capability.

One of the most powerful of the new features is a set of 40 Control Operations that enable the user to control the printer in a variety of 64-character alphanumeric formats

and list programs (including mnemonics), data, and labels; calculate a few special statistical functions; control and announce partitioning; configure error-state processing; and control increment/decrement of the first ten data registers. Control operations are executable manually and under program (user or CROM) control.

The 59's mag cards are slightly smaller than the 52's, and so cannot be used interchangeably. Each card reads/writes 2 of four memory banks, irrespective of partitioning. Each bank holds 30 data registers or 240 program steps, or equivalent mixes of each. Data registers number from the last of bank 4 up through the last third of bank 1 (2-digit addressing does not reach past Reg 99); program steps start at the beginning of bank 1, with step 959 ending bank 4. As examples: program steps 160-167 are equivalent to data register 99; steps 472-479/Reg 60; and 952-959/Beg 00. Partitioning is specified by data registers. For example, the sequence: 2 *Op 17 produces a 20/800 data/program-step mix. But since program protection is by bank, a bank that contains any/data registers cannot be protected, and steps 720-791 in this example could not be protected.

The 59 keyboard is the 52's 5 X 9 matrix with some rearrangement, renaming, and the following changes: x!, D/R, iferr, ifpos, ifzro, and $\sqrt[x]{y}$ have been eliminated, and CP, Pgm, x:t, Eng, Int, abs, Pause, x=t, Nop, Op, x≥t, ∑+, x bar, Rad, and Grad added. Most key codes follow the usual quasi-x,y addressing, with a few exceptions to handle merged Ind functions. 8 of the possible 100 op-codes are unused, and are the 59's pseudos. Cursory investigation reveals that of the 8 (op-codes: 21, 26, 31, 41, 46, 51, 56, and 82), p82 is the most interesting: no matter what is displayed, it always causes the next step to be skipped; if zero (or in some cases a small number) is displayed, when the skipped step is R/S, Nop, or INVSBR (the 59's rtn), execution of the p82 sets an error condition. P21, p26, p31, and p41 (SST) appear to behave as they do for the 52. So it seems that there will be some challenge to exploring the 59's pseudos, if not so much as for the 52. Incidently, the 59's pseudos are easier to create because of a register-address merging feature: in LRN mode, any 2-digits following a keyed register command (i.e. STO, RCL, etc) are merged into a single step, and the STO, RCL, ... can be deleted later. (You can do the same thing with GTO). But when you're editing a register sequence you have to rewrite the command as well as the address. For example, if you wanted to change STO 73 to STO 74, starting at the step containing tho 74 would produce the 2 steps: 07 04 instead of the desired 74, so instead, beginning at the STO step: STO 74 would give you the desired sequence.

Other non-52 or 56 features include: 6-level subroutine calls; 10 flags; indirect flag, Pgm, Op, fix addressing; Dsz of Reg 00-09; SBR n from the keyboard initiates execution (true for the 57 also); 13-digit display arithmetic; and a manually initiated R/S during program execution will not halt execution in the middle of connected sequences.

On the negative side, varying in degree of severity and subjective importance: Only 8 mantissa digits are displayed when in Sci or Eng format (like the HP-25), although 13 digits can be carried in the display register unless EE is executed, in which case rounding is to 8 places; there is no built-in factorial function and although Pgm 16 (Combinations, permutations, factorials) will calculate a factorial, 3 subroutine calls are required, and Reg 01-04 are used; polar/rectangular functions and character-print buffers use 4 stack registers; there is no manual D/R switch or other equivalent to the

52's-interrupt processing capability; there is apparently no user access to the stack registers, and thus no fractured digits; the same 52/56 trig and log function anomalies prevail (V1N2p3 and V2N2p1); the same code-transfer rules apply (V1N2p2); there is no INV or Ind viability through subroutine calls; and a timing comparison of a short Dsz loop showed that the 59 took twice as long as the 52 (varying display format appeared to have ne effect). This longer Dsz execution time is probably due to the addressing of a particular register (which neither the 52 nor 56 have to do).

The following features will probably be viewed by some users as plus and by others as minus: Program execution halts when an undefined label is encountered; mag cards cannot be protected from accidental over-write (although since read is automatic there should be little need for the black tab type of protection); there appears to be no special 0 divide error state (V1N1p2); and there are no "crashes" except for the p21 sin sequence (see the pseudo table elsewhere in this issue).

One of the powerful features of the 59/PC-100A combination is that they can be programmed to operate like an interactive computer terminal. The following program illustrates this capability. Press A, and you're off on a graded course in arithmetic!

TI-59 Program: Interactive Arithmetic Teacher Ed

Program Listing:

```
000:  *Lbl B STO 0 *Lbl E RCL*Ind 0 *Op 1 *Op 20 RCL*Ind 0 *Op 2 *Op 20
018:  RCL*Ind 0 *Op 3 *Op 20 RCL*Ind 0 *Op 4 *Op 20 *Op 5 INVSBR
033:  *Lbl C *CP *B' X RCL 79 = *Int *x=t C STO 75 *Lbl 3' *B' X RCL 79
052:  = *Int *x=t 3' x:t RCL 75 INVSBR *Lbl D *Pause x:t *Pause INVSBR
066:  *Lbl A 1 B E E E E E *Adv E *Adv *Lbl 9' 10 STO 79 0 STO 77 STO 78
089:  *Lbl 1' 29 B 0 R/S *CP *x=t 2' C + D STO 76 = x:t 47 *Op 4
110:  *Lbl 6' CLR R/S STO 74 *x=t 4' *A' 313237 *Op 4 x:t *Op 06 453241
136:  *Op 4 RCL 74 *Op 06 53 B *Adv GTO *1' *Lbl *2' E 0 R/S *CP *x=t
155:  *5' C + x:t = STO 75 - D STO 76 = x:t 20 *Op 4 GTO *6' *Lbl *5'
176:  E 0 R/S *CP *x=t *7' C X D STO 76 = x:t 50 *Op 4 GTO *6' *Lbl *7'
197:  E 0 R/S *CP *x=t *8' C X x:t = STO 75 div D STO 76 = x:t 72
217:  *Op 4 GT0* 6' *Lbl *8' *Adv 57 B 3221 *Op 4 *RCL 77 *Op 6 RCL 78
239:  *Prt E E 0 R/S *CP INV *x=t *9' E *Adv *Adv *Adv CLR R/S *Lbl *4'
256:  1 SUM 77 *A' 3632 *Op 4 x:t *Op 06 45 B E 0 R/S *CP *x=t *1' 10
280:  *Prd 79 GTO *1' *Lbl *A' 1 SUM 78 RCL 75 *Op 06 64 *Op 4 RCL 76
299:  *Op 6 INVSBR *Lbl *B' *pi x RCL 73 = INV *Int STO 73 INVSBR
```

Pre-stored Data:

```
01:  2324572465  3000223224 3122003732 37173637 4532413500  1335243723
07:  3017372415 0 1314242724 3745401331 3643173500 3441173620 3724323136
14:  2002132 3500451736 5700010000 2132350031 140273232 2600213235
20:  16243620 3327134520 3313413617 1600323317 3513311636 2617450013
26:  3136431735 5737231731 35413140 4313313700 3732001316 1671000000
32:  0 4313313700 3732003641 1437351315 3771000000 4313313700 3732003041
39:  2737243327 4571000000 4313313700 3732001624 4224161771 0 4532410043
46:  1735170035 2422233773 43133137 2313351617 3500333532 1427173036
52:  7100000000 1424351620 1435132431 1716001641 3115177300 4532413500
58:  3615323517 24360000 0 3335321427 1730364043 1331370037 3200373545
65:  2132350013 14173737 1735003615 3235177100 2313421700 1300312415
71:  1700161345 4014451740 .1415926536 0 0 0 0 0 0
```

While you probably won't be able to try out this program for awhile (the first production 58/59 machines aren't expected to-be on retailers' shelves until late June or early July), at least one retailer is selling the owner's manual now (Lectro-Media Ltd Box 1770 Philadelphia, PA 19105 (800) 523-2906, for $12.95) ... a 250-page 8½ X 11 combination programming guide and functional analysis which covers a lot of ground, and can get you started writing programs and deciphering mine (and/or improving it) while you're waiting to get a 58 or 59.

Although 52-NOTES will be giving the new machines increasing coverage, some space will continue to be devoted to significant topics concerning the old, depending upon member interest; important functional and operational comparisons of one machine with another will be of general interest for some time to come; then, of course, the 58 and 59 will have to wait for HP to catch up before they can fairly enter the Friendly Competion arena!

Pseudo Behavior Summary (52)

Herewith a quick reference for pseudos, which consolidates inputs from many members;

- 21: A, E, F; causes crash* when followed by sin, cos, tan, P/R, or D.MS during program execution
- 26: A, B, D, F;
- 31: D; causes a halt in LRN mode during program execution
- 61: A, B, D;
- 62: B, C, D;
- 63: D; executes as iferr
- 64: D; executes as INV
- 66: A, B, D;
- 71: A, E; SST of the first of a string of n p71s causes a skip of n+1 steps
- 72: B, C, D; cannot be inversed
- 73: D; executes as rset
- 74: D; executes as GTO
- 76: A, B, D;
- 82: B, C, D; cannot be inversed
- 83: D; causes complex interaction among display, arithmetic stack, internal registers, and functional states
- 84: D; similar to p83
- 92: B, C, D;

A=conditionally neutral; B=can provide missing operand; C=executes as EE and softens display selectively; D=may be used as a label; E=executes as a no-operation; F=when SST'd, creates a pending shift;

*Machine displays the two minus signs, and will not respond to any keyboard command except power off.

Hardware Modifications

Bob Edelen (100), Michael Rak (502), and Tom Scogin (517) are emerging as the Club's top hardware modification enthusiasts. Both Bob and Mike are starting chip add-on services, and expect to do other machine mods for interested users. I will publish the names of other Club mem- bers wishing to announce services related to PPCs and their use, but in no case should such announcements be construed as carrying Club affiliation or endorsement. Incidently, members who wrote to Mike (V2N5p3) may wish to contact him again for more details; both he and Bob report special mounting requirements for SR-52As.

More on Corner the Lady (Wythoff's NIM) (52)

Michael Brown (128), Dix Fulton (83), and Larry Mayhew (145) have written SR-52 programs for this game (V2N4p3), each taking a different approach. Michael's handles the largest playing field (1D10 X 1D10) but follows a partially iterative algorithm that costs execution time. Dix's is limited to a 99 x 99 field and also requires some trial-and-error search. Larry devised a non-Fibonacci Notation closed-form algorithm, and his program runs the fastest, handling a playing field up to 99999 X 99999. All three programs are well written, and it would be difficult to decide which represents the best programming, but I would judge Larry's to be the most playable. Larry modestly suggests that his algorithm is probably not as mathematically elegant as the Fibonacci Notation one, but there is no denying the proof of his pudding! While 52-NOTES is not the appropriate forum to pursue mathematical games in great detail, I will devote some future space to this topic should member interest/effort be sufficient. In the meantime, interested members might wish to contact Mike, Dix or Larry for further details of their programs.

A Few Tips on Presenting Your Inputs to 52-Notes

It will be easier for me to use your inputs to 52-NOTES if 1) separate topics are on separate pieces of paper, 2) little or no text rewording is required, and 3) programs are typed following a modified Dix Fulton format (V1N3p4), i.e. mnemonic strings with row-wise catenation (filled lines), without omitting the * (for 2nd) symbol. Incidently, although several have suggested the / symbol to denote division, in certain contexts it might be misinterpreted as a delimiter, and I prefer to stick with div. Try to use mnemonics as close to keyboard inscriptions as can be typed, follow upper and lower case conventions, and run merged code together. Note that for the new machines (58 and 59), LBL is Lbl, and rtn is INVSBR. I suggest that we adopt the symbols: abs for absolute value, S+ for sigma plus, and use 0', 1', ... 9' for labels, rather than the actual shift functions (Dsz, ifflg, ... Op) they represent. I am now inclined to reverse an earlier position (V1N1p5) and suggest that prograns submitted for 52-NOTES publication be in relocatable form to make them easier to modify for individual preference (the SR-56 excepted).

SR-52 Program Review: Analytic Computer Model by Ron Zussman (88), Computer Design, May 1977 pp105-109; reprints @$1.50 ea or entire manuscript @$3.00 ea available from Ron. As big and fast as modern computer operating systems are, their limitations are finite, and their efficiency depends in large part on how well hardware resources have been configured to match specific job loadings. Analytic models have been devised to determine optimum mixes and arrangements of computing machinery for predicted user traffic. Ron's program mechanizes one of these, based on queuing theory. From 7 inputs: 1) Time the central processing unit(s) is (are) active (CPU(s)) for each job, 2) proportion of (1) used for I/O, 3) average service time for peri- feral storage access, 4) total number of jobs, 5) number of system users, 6) Average wait times (for interaction), and 7) the number of CPUs, this program calculates CPU utilization, thruput, and response time for both exponential and constant models. Members applying this program to real operating systems are invited to convey comments/ results to Ron and/or me. Users should note that the user instructions in the CD article incorrectly list results under a "Press" column (p 107). Corresponding keys may be found at the bottom of p 109.

Tips

A y^x Workaround: In cases where you want y^x with x already displayed, Roy Grubb (483) suggests: X y lnx = INV lnx which takes advantage of the mathematical identity: log $y^x \equiv$ xlog y.

A Decrement Only dsz (56): Roy Chardon (515) notes that ... *dsz CE ... decrements Reg 0 without causing an error condition (analogous to the V2N2p1 sequence for the SR-52).

0 divide Error State (56): Roy also notes that several functions are affected during a 0 div error state for the 56: notably, the statistical and rectangular/polar; also that PROD executes as INV PROD (contrary to Dave Johnston's finding (V1N2p6)).

TI Notes

The customer service toll-free number (V1N1p6) should be changed to 800-858-1802; use 806-747-3841 (not toll-free) for technical assistance.

For those of you going to Dallas-for the NCC (13-16 June), TI expects to have several applications people there to answer questions concerning the new machines, and if they're not too swamped, you might be able to get your hands on a 57, 58, or 59. The Professional Calculator Division is in the process of moving to Lubbock, so just who will be where and when at the NCC can best be found out by contacting TI at the Dallas Convention Center when you get there.

TI is starting a separate program exghange for the 59 (PPX-59), with its own newsletter and catalog. More details as they become available. There are no plans to drop PPX-52.

Coping with Built-in Function Anomalies

Since most built-in functions produce approximations to desired results, and since the accuracy of the approximations is often dependent upon the magnitude of inputs, it is not surprising that some data regions cause problems. Manufacturers are motivated to find a compromise between complicated algorithms that handle all cases with equal accuracy and simple ones that break down significantly for specific inputs. It appears that HP considers it cost effective to spend more for "cleaner" functions than TI does, although the best algorithms don't necessarily cost the most to implement. Seemingly unrelated machine architecture features can affect the accuracy of results. For example, HP-machines do not flag underflow conditions, which makes it easier to process data whose magnitude is close to zero. Joel Pitcairn (514) notes that knowledge of algorithm construction would enable users to identify critical data regions, but unfortunately the manufacturers consider such information to be proprietary. A TI spokesman points out that even publishing information identifying critical data regions would give the competition useful ammunition. So I suspect the best approach is to find out as much as you can about the way a machine processes problems of interest before you buy, and then for significant programs, devise test procedures that cover all expected data ranges. And then ofcourse, when in doubt, round! For you mathematicians, Joel and Barbara Osofsky (420) both note that lnx of numbers in the neighborhood of 1.0000007 results in only 6-place accuracy. Joel also finds similar accuracy degradation for sin .00007 and tan 1.0000000002. Incidently, the 58 and 59's built-in math functions appear to be the same as for the 52 and 56.

Volume 2 Number 7 48/39 July 1977

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

A Few Observations on the Occasion of our First Anniversary

While most of you seem to be pleased with 52-NOTES subject matter and technical level, some have expressed a sense of difficulty in following a few of the more detailed discussions, and have suggested that a more elementary approach would be better. My aim has been to consolidate and present input material in such a way as to reach PPC users of above average intelligence, inquisitiveness, and attention span, but who may not have pursued formal education past highschool. As I have already suggested to a few, if after careful perusal of specific articles, having worked (actually run) all examples, and carefully referred to cited earlier issues of 52-NOTES you still don't understand something, write me, and I'll do my best to help. While the college professors among you have contributed significantly to 52-NOTES content, so have others with considerably less formal education. In order to make 52-NOTES as attractive to all concerned as I can, and keep the size of each issue within pleasantly managable bounds, I have pursued comprehensiveness with brevity as a general approach.

I expect to continue to give priority to software inventions and discoveries that have potentially broad application. Non-TI-approved hardware modifications are apt to be risky, and will not often be discussed in 52-NOTES. Even when mods appear to work, insidious problems such as power supply overloads and potentially dangerous (to hardware) instruction sequences may cause serious trouble. I intend to continue sharing the best of my own discoveries and inventions covering all the TI PPCs, so long as other members continue to do the same. The ability to "hide" 59 programs on protected cards will motivate some to try to market their best programs. I will publish only the names of members who have such programs to sell; interested members can write to the individuals concerned for further details. But I hope that most will be motivated to share their best ideas openly.

For the 58 and 59, a particularly rich and rewarding field to explore is the efficient use of CROM code. There are apt to be many subroutines, or portions thereof, that can be cleverly used in applipations other than those for which they were written. The sequence: *Pgm nn SBR mmm will call a code sequence in Program nn beginning at step mmm, and ending with the first INVSBR encountered (a feature not noted in the owner's manual). Since every 58 or 59 user has the Master Library module, clever use of its code can be taken advantage of by all users. As special applications modules become available, they too can be tapped for clever, efficient use by those who have them.

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 Vanderburgh 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.

The program that follows this article was designed to enhance the ML-02 matrix program, facilitating inputs, labeling outputs, and it makes use of the *Pgm nn SBR mmm sequence.

For those of you who would like to explore topics applicable to all (or most) of the TI PPCs, there are still some outstanding invitations to the membership: approaches to teaching computer programming (V1N2p1), specific requests for help (V1N2p4), printer techniques (V1N3p1), rigorous diagnostic programs (V1N4p4), systematic exploration of pseudo behavior (V1N5p3), comprehensive determination of all factors affecting program execution time (V1N7p3), reviews of periodicals of interest to PPC users (V2N2p4), application program generation input (V2N2p5), random number generators (V2N3p2), and program challenges to HP users (V2N4p2).

I'll close this commentary by repeating what I said some months ago: *"... it is with pleasant anticipation that I look forward to getting inputs from members who have so far been among the silent majority, as well as to getting more gems from the productive minority."*

TI-58/59/PC-100A Program: Enhanced ML-02 Program Ed

User Instructions:

1. Key n, press A, see 8 displayed (address of first storage register); n and N printed.
2. Key ith element, with columnwise catenation, press R/S; e_i printed, address of next register displayed; repeat for i=1,2,...n^2; correct entries by direct STO (display-cued); determinant calculated and printed following input of e_n2.
3. For simultaneous equations: If Det ≠ 0, press B, "B" printed; key b_i, press R/S; b_i printed; repeat for i=1,2,...n; repeat step 3 for new constant vector; x_i calculated and printed (with labels) following input of b_n.
4. For inverse, press C; inverse printed and labeled by column.

Note: For TI-59, max n=8; for TI-58, max n=3, and change all references to Reg 89 to Reg 29.

Program Listing:

```
000:  *Lbl D x:t 1 SUM 89 RCL 89 INV *x=t *5' + 2 = STO 89 *Lbl *5'
018:  *Op 04 INVSBR *Lbl E R/S *Op 06 STO*Ind 01 *Lbl *1' *Op 21 RCL 01
034:  R/S STO*Ind 01 *Prt RCL 01 INV *x=t *1' *Adv *Adv INVSBR *Lbl A
048:  STO 07 STO 00 x² + 7 = x:t 31 *Op 04 RCL 07 *Op 06 *Adv 13 *Op 04
070:  8 STO 01 E *Op 00 161737 *Op 03 *Op 05 *Pgm 02 C *Adv *Adv R/S
092:  *Lbl *B 1 *Pgm 02 D RCL 07 STO 06 *Op 00 14 *Op 02 *Op 05 CLR
111:  *Lbl *7' R/S *Pgm 02 SBR 355 *Dsz 6 *7' CLR *Pgm 02 E *Pgm 02 *A'
129:  4402 STO 89 *Op 04 RCL 07 STO 06 *Lbl *6' RCL*Ind 01 *0p 06
147:  *Op 21 4408 D *Dsz 6 *6' *Adv *Adv *Adv CLR R/S *Lbl C CLR *Pgm
166:  02 *B' *Op 00 243142 *Op 03 *Op 05 *Adv 1502 STO 89 *0p 04 1
190:  *Lbl *2' *Pgm 02 *C' RCL 07 STO 06 *Lbl *3' *Op 21 *Op 24
205:  RCL*Ind 01 *Op 06 *Op 36 *Lbl *4' *Pgm 02 SBR 860 *Dsz 6 *4'
221:  *Adv 1508 D 1 + RCL 03 = *Dsz 0 *2' *Adv *Adv *Adv CLR R/S
```

0^x Definitions

G E Wilkins (25) notes that the various PPCs produce varying results upon execution of 0^x with the y^x function, as the following table shows:

Machine	0^0	0^{+x}	0^{-x}
SR-52	1	0	0
SR-56	1	0	0
TI-58/59	1	0	error
HP-25	error	error	error
HP-55	error	0	error
HP-65	error	error	error
HP-67	error	0	error

Perhaps this is not too surprising, since there appears to be no clear cut mathematical approach to defining 0^x. G E suggests that one can argue that since log y^x = xlogy, and log 0 is mathematically undefined, then 0^x should also be undefined. Or on the other hand, one can examine the limit of y^x as x and y approach zero, and find that 0^0 ought to be 1, and 0^x ($x \neq 0$) ought to be zero. In any case, G E is looking for an efficient SR-52 routine that treats y^x such that an error is produced only when y is negative and x is not an integer, or when y is zero. He wants a correct result with correct sign when y is a negative real, with x any integer or 0, or y is a positive real with x any real, all in less than the 65 steps it has taken him. Members are invited to help out, and/or shed more light on the 0^x definition problem.

Hyperbolic Functions Shortcut

Roy Grubb (483) passes along the following approach to producing hyperbolic trig functions, which he came across in a Sinclair Electronics manual for its small PPC: ***LBL E** *(((INV lnx + 1) 1/x X 2 - 1) +/- INV tan X 2) *rtn*. Using this basic routine, the desired functions are produced by: sinh: E tan; cosh; E cos 1/x; tanh; E sin; sech: E cos; cosech: E tan 1/x; coth: B sin 1/x. Inputs should be in radians; any angular mode will do.

More on Reg 60, 61, ...69 Behavior (52)

In attempting to find a pointer-filling shortcut, Phil Sturmfels (49) discovered a means of sensing which values held in the pending arithmetic stack are "non-normalized". (Following a convention adopted by the HP-65 Users Club, I will use the term "normalized" to apply to numbers composed only of binary-converted-to-decimal (BCD) bytes; "non-normalized" to numbers with one or more of their 16 bytes representing hexadecimal (base sixteen) numbers.) The only way to move the contents of Reg 60 to a program register for detailed examination is through the display register, which always normalizes a non-normalized number (see V2N1p4). Register arithmetic on a non-normalized value in Reg 60 also appears to produce normalized results. However, as Phil discovered, a non-normalized number in Reg 60 is not affected by its use for indirect addressing, and its pointer behavior can identify it as non-normalized. For example, key: 1 + 2 X 3 y^x in RUN mode. Find that Reg 60, 61, and 62 appear to contain 1, 2, and 3 respectively. Now use these registers as pointers, and find that only the 2 behaves as a 2; the "1" and the "3" behave as 0, and it appears that they are non-normalized. It may be that there are some non-normalized numbers that point normally, and this phenomenon probably deserves further exploration.

Pseudo 73 Limitations (52)

Larry Mayhew (145) points out that p73 *"... cannot be trusted as the equivalent of rset without the flag modifications, because under certain important conditions it will simply be ignored during program execution. Example: Beginning at step 000, write: HLT INV *ifflg 1 *1'* ***LBL *1'*** *12 p73 45 HLT. In RUN mode key *rset RUN RUN and see 12 as expected. Now key *rset *stflg 1 RUN RUN and see 1245, which shows that the p73 executed as a no op. In general, if any conditional test has been made without a transfer occurring, and if after that test p73 occurs and there has been no intervening STO, RCL, SUM, EXC, PROD, CLR or fix, the p73 will be ignored."*

Step 223 Odd Behavior (52)

Larry also notes that "If step 223 contains part of a numerical address (program or register) that is left incomplete at 223, and if the program runs through step 223, and a digit is then keyed manually, a 'semi-crash' will occur: Every register is cleared, but flags remain unchanged." Or, if a conditional instruction is positioned at step 223, and executed with RUN n RUN where n is a digit, the machine goes into a read state. Larry has also gotten a step 224 to appear in the display in a sort of through-the-lookingglass Alice-In-Wonderlandish hocus pocus by way of a run-down battery. Write him for details.

Information Referrals

Largely to save time, I have adopted a policy of suggesting via 52-NOTES that members contact a contributor directly, without my having obtained prior approval from him in cases where I consider topics inappropriate for 52-NOTES coverage. I don't believe this has caused contributors to be overwhelmed with solicitations, but it always helps to include at least a SASE with each request. Members not wishing such announcements to be made should so indicate when contributing material.

More on Don Ellis' Publications (V2N3p3**)**

B.K. Patterson, Jr (272) has examined some of Don's statistics programs and reports an error in one of them. Members using Don's publications may wish to contact B K and/or Don for details. But send B.K. some change (rather than a SASE) since US stamps won't be of much use to him in Canada.

More on Sum-of-the-Digits (V2N5p6)

I've received a number of 13-digit sum routines now, most of which won't handle all numbers, which makes them hard to compare. So here is a new routine from Howard Cook that appears to handle all reals, but requires 56 steps (Karl Hoppe (507) has one with 61 steps). Shorten it if you can, but "better" routines must be able to handle all reals.

```
*LBL E INV *fix *ifpos +/- +/- *LBL +/- STO 01 EE div EE 00 = INV *Prod
01 *fix 0 *LBL *LBL CLR RCL 01 *ifzro *EXC - .5 = EE SUM 99 INV SUM 01
10 *PROD 01 GTO *LBL *EXC *EXC 99 *rtn.
```

Comparable routines for the 56/57/58/59 ought be be shorter.

Decapower Zero Suppression Following Negative Tests (52)

Jared Weinberger (221) notes that any decapower zero is suppressed following an unmet test using a defined label for its transfer instruction. The suppressed-zero display can only be seen when a test is executed manually or SST'd, since a HLT or rtn restores the suppressed zero.

SR-56 Program Exchange Update (V1N6p6)

Dave Johnston's 1 July 77 Catalog lists 86 programs covering math, statistics, physics, games, finance, operations research, and misc. Dave plans to get a TI-57 and will broaden his exchange service to include 57 programs. And since TI does not plan to provide a program exchange service for the 58 either, there will be a need to be filled for that machine too. Write to Dave if you plan to buy either the 57 or 58. If response looks like more than he wants to handle, we'll look for additional help. Those members with 58s who join TI's PPX-59 and wish to contribute programs to it may send me copies of programs on TI forms with sample problems, all checked out on the 58, but scaled to the 59, and I will put them on mag cards (until such time as there are too many such requests). Include blank mag cards and a large SASE.

Machine Coverage in 52-Notes

Since the nature of contributed material influences how I prioritize newsletter topics, if you want to see more coverage of a particular machine, write about it; clever routines, inventions, and discoveries are what I look for most, and appear to be what most of you prefer too.

Data Entry Sensing

Izzy Nelken (576) asks if there is a way during SR-52 program execution to sense whether data have been entered from the keyboard. There is no built-in function in the TI machines comparable to the HP-67 Flag 3 (which is automatically set and post-test cleared when data are entered during a program halt or pause), so it appears that the display would need to be examined, assuming that keyed data would fall in a given numerical range outside of the possible display values produced by the program. If anyone has a better idea, Izzy would be pleased to hear from you. But it's probably more practicable to design programs to always expect some data to be keyed at specific break points. A keyed zero or one provides easily tested data for a two-way branch: a handy way to respond yes or no to printed questions (see the TI-59/PC-100A program in V2N6p3).

Membership List Corrections

44: Box 233; 97: 18 Carty Ave Ft Nonmouth NJ 07703; 307: Collins Radio Group MS 424-101 Dallas, TX 75207; 368: 2631 Navarre Ave #209 Oregon OH 43616; 376: Buchenweg 24 D-5000 Koeln 40 Germany; 518: Math Dept SUNY Geneseo, NY 14454.

Assessment of the 58/59 Master Library CROM Programs

Members with appropriate expertise can contribute significantly by assessing/analysing specific ML programs. All can be downloaded into 59 user memory; all but ML02 and ML19 into a 58's. If you have a 58 but not a 59, and want listings of these two programs, send me a SASE. A cursory look at ML-02 (the longest CROM program) shows a repeated use of in-line code sequences, which may be justified by increased execution speed. But it also appears that more data registers than necessary are used. I invite more analysis of this and other ML programs, and will air pertinent discussion via 52-NOTES. While we can't change the CROM code, we can use it more effectively if we understand its structure and limitations.

58/59 Tips

Short Form Addressing: Although the omission of leading zeros when keying register or step addresses can save manual keystrokes, it is easy to forget that a non-digit keystreke must follow. So it can be worth the extra manual steps to avoid address mistakes that can be time consuming to find; the number of required program steps is the same either way.

ML-01 Print Routine Use: The last sentence in the user instructions should read *"... except that the program **must** not be called.*" If a program other than ML-1 is accessed, automatic printing won't occur. Also, only manually keyed user defined keys (A-E') will initiate the automatic prints (calls to other labels, absolute addresses, and/or R/S sequences will not). This routine does make use of indirect program accessing, and is worth examining (via Op 9 dowmload) to see how it works.

Taking Advantage of Control Operations: From time to time, a glance at page V-27 of the owner's manual will remind you of all the things the 40 special control operations can do. Keep in mind the use of Ops 20-39 in place of programmed 1 SUM n or 1 INV SUM n (where 0 < n < 10); they save steps, don't alter the display, and run faster.

INV and Ind Combinations: There are a lot, and some can be complicated. Page V-68 sorts them out, showing proper sequences, and which Ind combinations are merged. One misprint that caught my eye is in the 19th entry: replace Dsz with ifflg. Remember that in all cases, Ind follows the associated direct instruction (unlike the 52 or 56). Note that for both flag and Dsz commands both the flag number or Dsz register and transfer address can be indirectly specified. A double indirect sequence such as: *Dsz *Ind 25 *Ind 26 must be completely rewritten if either indirect address is to be changed, otherwise the paired digits would not be properly merged. Incidently, *Dsz *Ind XX will dsz the contents of any addressable data register within the current partition as specified by the contents of Reg XX (not just Reg 0-9 as is the case for direct Dsz). For direct Dsz it is important to remember to key only a sigle-digit register address. For example, *Dsz 0 4 keyed in LRN node is interpreted as *Dsz 0 004, which means decrement Reg 0 and go to step 004 if the contents of Reg 0 is not zero.

The Nop Instruction: Contrary to what the manual says on page V-51, there are quite a few code sequences which an interposed Nop will alter, notably between a Command and an address, and between merged commands.

Getting Merged Code into the Last Partitioned Step: The mechanism by which some instructions are merged won't work at the last partitioned step. This appears to be because the first keystroke(s) of some merged instructions increment(s) the program pointer, and when this occurs at the last step, there is the automatic switch to RUN mode. The most likely such merged instruction to be at the last step is INVSBR (code 92), and here are two ways to get it there: 1) starting at the next to the last step, key RCL 92, then overwrite the RCL with the intended next-to-last instruction, or 2) if the last step is less than 959 (479 for the 58) temporarily partition to a larger program field, write INV SBR at the desired step, then repartition as desired.

```
*****    ***         *     *   ****   *****  *****   ***
*       *   *        **    *  *    *    *    *      *   *
*           *        * *   *  *    *    *    *      *
****       *    ****  *  *  *  *    *    *    ***     ***
    *     *           *   * *  *    *    *    *          *
    *    *            *    **  *    *    *    *      *   *
****    *****        *     *   ****     *    ******  ***
```

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 B & 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
095:  *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'
```

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 copdition 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 Partitioning: Be sure to mark the prevailing partitioning on a recorded nag 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>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 of course 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 char- acters 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 INV SBR 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 < 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 x:t 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 x:t RCL 57 x≥t 195 x:t
189:  STO 57 x:t GTO 204 RCL 59 INV *x≥t 204 x:t 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)

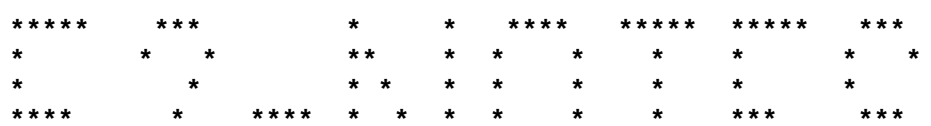
Dallas Egbert (384) offers the following discovery as a trivia award candidate: key *read, then simultaneously press the keys: B, INV, sin, STO, EE, 4, and 0. If you've done this correctly, the drive motor should turn on! I wonder if the SR-52 designers have a plausible explanation? Perhaps Dallas' discovery 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 ifflg, 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.



Volume 2 Number 9 48/39 September 1977

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

HIR Operations: A Major Discovery (58/59)

Heinrich Schnepf (376), who publishes the German PPC newsletter **Display**, discovered that p82 can be used to access the 8 pending arithmetic registers. TI acknowledges awareness of this capability, but has no comment when asked why it was not announced. A PC-100A listing gives p82 the mnemonic: HIR, which I interpret to stand for Hierarchy Internal Register. I will henceforth use HIR mn to refer to an operation, and HIR n to refer to HIR number n. Heinrich found that the sequence: HIR mn does the following: With n taking on the address of one of the 8 HIRs (n=1,2,...8), m=0 produces STO, m=1: RCL, m=3: SUM, m=4: Prd, m=5: INV SUM, and m=6,7,8 or 9: INV Prd. Nested arithmetic operations push operands first into HIR 1, then on into 2,3,...8; print buffering assigns HIR 5 to Op 1, HIR 6 to Op 2, HIR 7 to Op 3, and HIR 8 to Op 4; INV P/R uses the first 2 available HIRs, and HIRs 7 and 8; P/R uses the first available, and 7 and 8; D.MS and INV D.MS use the first 2 available, and 8 ... which hold potentially useful separations of some of the D.MS'd elements; sigma + and - use 7 and 8; x (x-bar) uses the first available; Op 11 uses the first 2 available; Ops 12 and 15 use the first 3 available; Op 13 uses the first 4 available; and Op 14 uses the first 3 available and 8.

Unlike the analogous SR-52 Reg 60-69, the HIRs do not reformat pushed operands, nor appear to attach operators to them. Thus operands used in nested arithmetic operations stay "clean" and may be retrieved intact with the appropriate HIR recall (HIR 1n); stacked operands may be changed (by HIR 0n) or modified (by HIR im, m=3,4,5,6) prior to operator execution. There is no (or doesn't seem to be a) fractured digits potential. Neither CLR nor CMs nor CP clears a HIR. The only way I've found that does (short of storing zeros or turning the machine off) is by Op 00, which only clears HIRs 5 through 8, and even then only if the 58/59 is plugged into the printer; without the printer, Ops 0-5 execute as Int.

HIRs 5-8 do reformat print code, and do not perform normal register arithmetic on print code contents produced by Ops 1-4. However, print code can be synthesized and put into HIRs 5-8 in such a way as to make possible normal register arithmetic. This leads to the ability to modify print code directly in a print buffer, obviating the need to use ordinary registers for such modifications. The printer appears to act upon the ten LSDs of the 13-digit mantissa or a print-buffer HIR: a situation similar to SR-52 fractured digits synthesis (V1N2p5).

Then synthesizing print code, it is necessary to fill the 3 MSDs with non-zero numerals, then follow these with the desired ten digits of print code. For example, to artificially get the printer to print ABCDE in the leftmost print sector, key 99913141 EE 5, store with HIR 05, key 51617, SUM with HIR 35. Now key Op 5, and see ABCDE printed. At this point, if you want to change the C to say an *, key 3.6 EE 5, and sum with HIR 35. Key Op 5 and see AB*DE. If you try to start off with 1314151617 Op 1, HIR 35 register summing won't work; anyone know why?

Register arithmetic in the HIRs only works for integer, full integer-fraction, or floating point operands. The implied decapower sign of a fixed point fraction gets changed from minus to plus (Position B changes from 4 or 6 to zero or 2 (see V1N1p5 and V1N4p5)).

User access to the HIRs will undoubtedly find many practical as well as esoteric applications. In an arithmetic teaching application, Heinrich has written a program that zeros HIR 1 prior to halting for the student's answer. Following the input answer, HIR 1 is tested for zero, as a cheating indicator. In a biorhythm program, he uses HIRs 2-8 for storage of intermediate results, freeing data registers for print code. This program also demonstrates the plotting of multiple points, and makes use of CROM program ML-20 to perform calendar calculations. Fine tuning the partitioning to an effective 463.61 saves otherwise wasted space, but requires repartitioning during program/data entry: 479.59 to key in the program, and 399.69 to prestore the last two print constants. The program that follows contains an English translation of Heinrich's German messages; run it by pressing A and following printed instructions. It appears that clever accessing of the ML-20 program via the SBR nnn approach (V2N7p1) and making appropriate register reassignments could save quite a few precall data formatting steps, and perhaps some of you will want to try this. With the listing of this program, I am trying out some new symbol conventions: The * symbol for 2nd is dropped, L=Lbl, S=STO, R=RCL, *=Ind; rtn=INVSBR, and groups of numerals requiring synthesis will be bracketted by the " symbol. For example, DSZ 56 182 is written: Dsz "56" 1 "82" indicating that the 56 and 82 must be artificially merged; p82 is written HIR which, of course, must be synthesized. At least one member has confused xEt with x=t, so I am substituting xXt for xEt. Leading zeros are omitted when address codes are followed by non-numbers.

Robert Snow (212) was probably within a few weeks of making Heinrich's discovery independently, having reported same to me today (1 Sept 77). Also today, a question from A B Winston (707) pointed to a use of the HIRs to provide a means for a running program to detect whether the 58/59 is plugged into the printer: Op 1 puts printer-formatted display into HIR 5 only if the 58/59 is plugged into the printer. The sequence: *... 0 HIR 05 7 Op 1 HIR 15 OP x=t 1'...* transfers to L1' if the printer is not connected, continues if it is. This would allow programs to be I/O-optimized for operation both with and without the printer... a feature that would especially enhance many CROM programs.

TI-59/PC-100A Program: Biorhythm Analysis H.Schnepf (376)/Ed

Program Listing:

```
000:  Lbl E STO7 rtn Lbl E' 4 Lbl D' STO6 Op 0 RCL*7 EE INV EE Op*6 Op 27
      Dsz6 014 Op5
029:  CLR rtn Lbl A' 1/x X RCL4 = INV Int X 360 = sin X 7 = fix 0 EE INV fix
054:  INV EE - 7 = div 5 = +/- rtn Lbl B' STO7 INV Int EE 1 INV log = SUM*7
077:  CLR rtn Lbl C' div 10 STO 42 STO 00 - INV Int Prd 00 + 1.2 SUM 00 = Int
      EE 2
103:  + RCL 00 INV x≥t 112 + 2 = INV EE rtn Lbl B 27 E E' E' INV stflg7 Adv
127:  rtn Lbl C 35 E 1 D' R/S Prt EE +/- 4 HIR 2 2 D' R/S Prt EE 2 HIR "32"
150:  1 D' R/S Prt + HIR 12 = STO 0 Adv rtn Lbl A 7 Op 17 Adv Adv 50 E E'
173:  Adv E' E' Adv 6 Op17 43 E 3 D' 2 D' C Pgm20 A Op19 INV ifflg7 202
198:  B GTO 180 2 D' RCL 00 Pgm 20 D STO42 x:t 4 INV x=t 222 8 E GTO 230 10
224:  SUM 42 RCL*42 STO 42 2 D' Adv 43 E 3 D' 2 SUM 07 2 D' C Pgm 20 B Pgm 20
      C
250:  Op19 ifflg 7 260 CP x≥t 264 B GTO 233 ST0 4 78 SUM 05 Adv 17 E 2 D'
276:  2 D' 2 D' Adv Adv E' RCL 5 - (365.25 X (1/x x (RCL5 - 122.1 )) Int)
310:  Int = STO 3 div RCL 14 = Int HIR 4 X RCL14 = Int INV SUM03 1 x:t HIR
      14 -
334:  13 = x≥t 344 + 12 = HIR4 10 x:t RCL3 C' EE 4 INV EE + HIR 14 C'
360:  + 2 EE 5 = STO3 CLR STO0 STO1 ST02 23 A' HIR8 x:t 33 A' HIR 6 28 A'
      HIR 07
390:  x=t 404 HIR 16 x=t 412 47 X HIR 18 B' HIR 17 x:t HIR 16 x=t
410:  418 50 X HIR 17 B' 51 X HIR 16 B' .24 x:t RCL1 EE div 6 INV log =
436:  INV Int x≥t 447 214 EE 4 SUM1 CLR E E' Op25 Op24 Dsz 42 361
458:  GTO 283
```

Prestored Data:

```
08:  3616134500 4317163117 3613374135 364131 303231 37411736 30.6001
15:  3723413536 213524 3624151327 4764332345 3713270000 5064301731
21:  3524376527 5164363324 3332364000 4000010000 311722 1613371700
27:  3033413717 6537001532 1632173631 1613371700 2213243140 24370013
33:  1700373545 3327171336 4517133520 7100000000 3032313723 1613457100
39:  13000000 2437002436 1613454000 364131 24310000 1700261745
45:  3327171336 1613371720 1424353723 16133717 3637133537 4536243651
51:  13311327 2345372330 5114243235 3723351717 3600243100 16133717
57:  1731371735 35633640 43243723 4017131523 3313353736
```

Printer Head Cleaning (PC-100/PC-100A)

Since TI reports that running one of its printer head cleaning programs with heavy paper cleans the head by heating the dot-matrix elements, it would appear that programs energizing the most dots would be best. The TI-58/59 program (page VI-12 of the owner's manual) prints repeated lines of 20 8s, which exercises only 340 of the 700 dots. A routine along the lines of: Lbl B Op1 Op2 Op3 Op4 Op5 rtn Lbl A 3232323232 B 7676767676 B 2424242424 B GTO A cycles through the symbols 0 (code 32), capital pi (code 76), and I (code 24) which together exercise all 700 dots, when run by pressing A, and stopped with R/S. With regular printing paper installed, this program may be used to check out proper functioning of all the print dots.

The comparable 52 and 56 programs can also be substantially improved. Here the apparent objective is to exercise all the dots that can possibly be energized by these machines. At any position a numeral can be created (2-12, 14-16, 18-19), all 35 dots should be exercised; one should try to find the minimum set of trace mnemonics that covers all position 20 possibilities. I'll publish the best SR-52 and SR-56 printer head cleaning routines you send in.

Magnetic Cards (59)

Exploring Mag Card Protection: Lou Cargile (625) has found that he can get a protected card to read without setting the protection flag by deliberately causing it to misread. The types of misreads that work may be machine-dependent; others trying this are invited to share results. In the meantime, a few questions from Jared Weinberger (221) bring up an important aspect of card protection not clearly covered in the owner's manual: reading any protected side of a card sets the protection flag which affects the whole machine until cleared with a manual CP or by switching the power off. For example, if the machine is partitioned "959." (no data registers) and Bank 1 recorded as protected on one side of a card, when that side is read, all of memory is protected. Repartitioning and/or p31 instructions encountered during program execution of protected code are ignored. However, any bank that is not fully partitioned for program memory can be copied normally on a separate card even though the protect flag has been set. The protect flag can then be cleared, the card read, and the code examined. For example, cycle your machine off-on, then key 10 Op17 LRN Lbl A 123 R/S LRN 999 S99 888 S98. Now key 1 +/- Write and feed a card through. Cycle off-on, key 10 Op17 CLR, and read the card just written; see -1 displayed. Press A, see 123; RCL99, see 999, R98, see 888. But press LRN, and the machine stays in RUN mode, and any attempts to repartition fail. Now key 1 Write and feed a blank cardside through; see 1 displayed. Cycle off-on, then key 10 Op17 CLR and read the card; see 1 displayed. Press A, and the RCLs, and find things the same as for the protected card. But now you can get into LRN mode and/or repartition. The first cardside was sort of partially protected: once read, the program part could not be examined in LRN mode, but it could be copied to a second cardside normally, which could then be read and examined normally. This partial protection situation also applies to all of memory, not just to the one bank that has been read from a protected card. For example, a 319.79 partition says that all of Bank 1 is program, Bank 2 is a mix of program and data, and Banks 3 and 4 are all data. A cardside protect recorded with this partitioning from any of the 4 banks, when subsequently read denies only Bank 1 from being recorded. A consequence of all this is that the read-in of one protected cardside, by virtue of its having set the protection flag and frozen the partitioning, predetermines the rules by which all of memory will behave, regardless of whether subsequent card-reads are by protected or unprotected sides.

Incidently, although resident code affected by a protected card-side read cannot be SST'd, you can access it selectively by SBR nnn. This may make it possible in some cases to synthesize protected code by observing results when SBR nnn is executed repeatedly for all nnn from the end of the program partition back up through step 000. Then, if you're lucky, you'll intercept a sequence like: ...GTO 182 A rtn, and a SBR call to the step containing the 82 will return with the contents of HIR 1 in the display!

Mis -reads -writes: Joel Rice (3) reports considerable difficulty getting cards to read properly. We may find that because of the greater information density, 59 card read/write won't be as reliable as the 52's. Others having serious read/write problems are invited to share them. I've found that clearing program and data registers before a read, or a write key-in appears to help, as does wiping each card with a clean lint-free cloth, which should also be used if after read or write you can't remove the card without getting a fingerprint on it.

Mag Card Read and Write Under Program Control: The owner's manual (VII-5) notes that mag cards can be read under program control, but doesn't say what happens when executing code is overwritten by a read, or that card write can also be effected under program control: a feature brought to my attention by Bob Moore (488).

It turns out that if a cardside destined for Bank m is read by an INV Write instruction being executed in Bank n, n≠m, execution resumes at the step following the INV Write after the card has been read. But if m=n, and this bank number precedes the INV Write, execution of the old code continues past the m INV Write until that part of the read has been completed. Then execution continues on into the new code at the step following where the old code last executed. This amounts to as much as a second or so of parallel processing: card read and code execution occurring simultaneously. Duration depends upon where the INV Write is located within its bank: the closer to the end of the bank, the longer the parallel processing.

SR-52 Cards for the TI-59: Mack Maloney (246) has found that SR-52 mag cards are the same thickness as the 59's (.008") and that if they are trimmed to the 59's card width (.635") they appear to read and write properly. The .025" extra length doesn't seem to matter. As more users try this, we should be able to determine whether there is any problem with TI-59 re-writes over SR-52 code, which may not erase all the old code (a reported problem for HP-67 users when attempting to use old HP-65 cards). To be safe, you can first demagnetize the SR-52 cards.

A PC-100A Typewriter (58/59)

Thomas Cox (9) suggests that "It would be interesting if someone could devise a useable routine for the TI-58/59 that would use the Rausch keyboard to generate the corresponding characters on the PC-100A." The following program does what Thomas has asked for, running somewhat like a slow typewriter, providing analogues of the horizontal position indicator, carriage return, end-of-line bell warning, and replaceable characters of real typewriters. This is the sort of program that is worth refining to cut execution time, since there are broad practical applications. So send me your better versions... perhaps TI could be persuaded to put the best one in a new CROM.

Incidently, the rather surprising coincidence (or did TI plan it that way?) that the last column number (20) is also the print code for the hyphen symbol saves a couple of keystrokes. For the 58, partition 159.39; for the 59 turn-on will do.

TI-58/59 Program: PC-100A Typewriter Ed

User Instructions:

1. Initialize: Press A, see 1 (ready for first letter)
2. Key ith letter via Rausch overlay (V1N3p2), see i=1; display flashes when i+1=20, in which case either Rausch-key last letter to fill line, or press E to end line with a hyphen. Repeat step 2 until message is complete. To end an incomplete line with blanks, follow last letter with R/S. To generate a non-letter character, key the 2-digit print code and press E (instead of Rausch keying).

TI-58/59 Program: PC-100A Typewriter Ed

Program Listing:

```
000:  Lbl A Op00 20 x:t 4 STO28 1 STO31 STO33 Lbl 2' 5 ST00 1 EE 8 STO30 CLR
      STO32 Lbl 1'
030:  RCL33 INV x=t 3' Op55 Lbl 3' R/S RCL32 Op*31 GTO 4' Lbl D + 9 Lbl C + 9
      = Lbl B
057:  STO29 CE RCL*29 Lbl E X RCL 30 = SUM 32 100 INV Prd 30 1 SUM 33 Dsz0 1'
      RCL 32
084:  Op*31 1 SUM 31 Dsz 28 2' Lbl 4' Op5 GTO A
```

Prestored Data:

```
01:  36 42 45 25 30 33 13 16 22 37 43 46 26 31 34 14 17 23 41 44 0 27
23:  32 35 15 21 24
```

Membership Address Changes

3: 230 W 107th St #6J New York, NY 10025; 48: 3611 Wyatt Dr Holiday, FL 33590; 75: Box 699 College Station, TX 77840; 281: 1502 42nd St #3 Brooklyn, NY 11219; 306: 4523½ Avocado St Los Angeles, CA 90027; 343: 1325 Quaker St Golden, CO 80401; 346: 37 Winding Way Madison, NJ 07940; 347: VQ-3 Box 65 FPO San Francisco 96637; 440: 148 Transmitter Rd Oak Harbor, WA 98277; 449: 807 Ruggles Hall Columbia Univ New York, NY 10027; 471: Box 461 Storrs, CT 06268; 584: 266 Main St #325 Windsor Locks, CT 06096; 587: 501 Loring Ave Los Angeles, CA 90024; 594: 45A Hillview Ave Rensselaer, NY 12144.

Miscellany

How the PPCs Work: Gene Werner (120) has kept a record of some 20 or so books and magazine articles published over the past few years covering various aspects of calculator and PPC design. Send Gene a SASE and a few stamps for his bibliography.

TI Trig Algorithms: Joel Pitcairn (514) reports having mechanized the CORDIC technique (see **Byte** Aug 77 p 142) for calculating the tangent function and comparing results with the SR-52 built-in tan function. For base ten and Max=13, Joel finds such close agreement, especially for critical values, as to conclude that this is the approach used by TI in the firmware. Execution time comparisons suggest that the sin and cos functions are derived from the tangent; it also appears that TI also uses the CORDIC algorithm to get arctan. Joel notes an error in the **Byte** article: arctan $x_o=z_i+x_i$, not just z_i. Joel's findings are likely to apply to the other TI PPCs as well.

Owner's Manual Versions (58/59): Mack Maloney (246) reports that there are currently 2 versions: 1014983-1 and -2 (look on the lower right corner of the back outside cover to see what yours is). The –2 reportedly corrects most of the -1 errors, and has its own addendum sheet (#1019286-4).

Numerical Solutions To Partial Differential Equations: Larry Gearhart (442) would like to get in touch with anyone pursuing numerical solutions to PDEs via one of the TI PPCs.

More on INV and Ind Viability (52): Claude Belanger (254) notes that while INV and Ind viability are maintained through direct subroutine calls (to A-E') (see V2N5p4), viability is lost if the INV or Ind is followed by GTO or SBR.

TI Notes: Check with your dealer or recent TI ads for details concerning the free Leisure Library CROM for all TI-58/59 owners. PPX-59 is expected to get under way 1 Oct 77 with an initial catalog covering CROM programs and conversions of "the top 200" PPX-52 programs.

```
*****    ***         *     *   ****   *****  *****   ***
*       *   *        **    *  *    *    *    *      *   *
*           *        * *   *  *    *    *    *      *
****       *    ****  *  *  *  *    *    *    ***     ***
    *     *           *   * *  *    *    *    *          *
    *    *            *    **  *    *    *    *      *   *
****    *****        *     *   ****     *    ******  ***
```

Volume 2 Number 10 48/39 October 1977

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

Friendly Competition

Dix Fulton (83) has responded to John Ball's challenge (V2N8p4) with an SR-56 satellite predictor program that processes inputs and creates its own constants (neither of which John's does) in addition to producing the same results. So now the challenge is reversed! However, whatever the final outcome, John gets the credit for the efficient use of the polar-rectangular functions that shortens the trig calculations (V1N4p5) as well as a clever use of the x_m (arithmetic mean) function that shortens two division calculations. Here is Dix' counter challenger:

SR-56 Program: Satellite Predictor — Dix Fulton (83)

User Instructions:

1. Enter Inputs: Key Nodal Longitude (degrees), - Observer Long, =, press RST R/S; key observer latitude, press R/S; key orbital period (minutes), press R/S; key inclination (degrees), press R/S; key height (miles), press R/S; key time from nodal crossing (minutes), press R/S; see Azimuth (degrees) displayed.
2. Get Range (miles): press R/S
3. Get Elevation (degrees): press R/S

Program Listing:

```
00:  STO1 R/S STO2 R/S div 1440 = STO7 R/S STO4 R/S STO6 R/S div 4 = STO5
     f(n) Mean
28:  sin x:t RCL4 f(n) P/R STO0 f(n) xm cos x:t +/- f(n) R/P + RCL5 + RCL1 =
50:  f(n) P/R +/- EXC0 f(n) R/P - RCL2 = f(n) P/R x:t EXC0 f(n) R/P EXC0
70:  f(n) R/P x:t RCL6 + 3958 = x:t f(n) P/R - 3958 = f(n) R/P EXC0 R/S
95:  x:t R/S EXC0 R/S
```

J R Merrill (693) has submitted the following Extended Precision Powers SR-52 Program as an HP-67 challenger:

SR-52 Program: Extended Precision Powers — J.R. Merrill (693)

User Instructions:

For y^x key y (0 < INT < 100), press A; key x (INT) press B; see first ten digits of results; press C for next ten digits, and repeat until display flashes. x must be sufficiently small that y^x is less than 10^{220}.

SR-52 Program: Extended Precision Powers J.R. Merrill (693)

Program Listing:

```
000:  Lbl A STO62 rtn Lbl B STO67 CMs 0 STO99 1 STO98 98 STO69 97 STO66 0
      STO68 1 SUM66
038:  *RCL66 X RCL62 + RCL68 = *STO66 div 1 EE 10 + 1 EE 11 - 1 EE 11 = STO68
      X
075:  1 +/- EE 10 = *SUM66 RCL66 - RCL69 = INV ifpos 034 1 +/- SUM 67 RCL68
      +/-
107:  ifpos 122 1 SUM69 R68 *SUM 69 RCL67 INV ifzro 025 RCL69 + 1 = STO97
139:  INV EE 1 +/- SUM97 *RCL97 HLT Lbl C RCL97 - 98 = INV ifzro 141 0 1/x 0
      HLT
```

Specific matrices have been found for which Hal Brown's HP-67 program (V2N8p3) gives incorrect results without warning, requiring the user to calculate the inverse of the inverse as a validity check. Hal has been developing improved versions of his program, including one that prints (with the HP-97), but has not yet eliminated the requirement for this validity check nor what can amount to an unacceptable number of manual row/column interchanges. It would appear that he will need to provide for more pivoting and/or treatment of near-zero numbers as zeros to eliminate these shortcomings... and there may not be sufficient HP-67 memory for a one-card program.

Routines

A Short Flag-Reversal (52): Jared Weinberger (221) has devised: ... ifflg 0 LBL LBL INV stflg 0 ... as an efficient way to alternate a flag's state each time this sequence is exercised.

More on Sum of the Digits (V2N7p4): Jared is back in the running with: *Lbl 1' (STO99 EE ÷ EE 00) INV SUM99 1 + RCL99 Lbl A INV ifpos LBL LBL +/- INV ifzro 1' = rtn* which in 33 steps handles all 13-place reals with a call to A.

Automatic Number Printer (58/59): R.G. Snow (212) has devised a sequence to convert up to a 5-digit positive integer into the equivalent print code. Following is his routine, revised to be relocatable:

```
Lbl A CP x=t 1' ÷ log Int STO8 Op28 INV log Lbl 2' + x:t 100 Prd 21 8 x≥t 1'
2 + Lbl 1' 1 - Int SUM21 = X 10 Dsz 8 2' CLR EXC21 Op*4 Op5 rtn.
```

With the desired print sector (1-4) stored in Reg 4 and the number to be translated in the display, this routine (A) prints the input number in the desired sector.

Factorials: For the SR-52, Larry Mayhew (145) notes that display-rounding of a positive real presents a true integer to the x! Function (there is no error condithn set).

It may be worth some effort to find the most efficient ways to get X! on the 58/59, just as with Int/frac routines for the SR-52 since there is no built-in X! function for the 58/59. ML-16 will do factorials, but user program access takes 9 steps, and 69! takes 20 seconds to execute. A closed solution using Stirling's formula which approximates n! by: $(2npi)^{½}\ X\ (n/e)^{n}$ might be better for large n. I invite efficient mechanizations of this formula and/or better approaches from the membership.

Special Case Routines: Larry Mayhew (145) points out the value of using efficient special-case routines when there is assurance that data involved are consistent with applicable restrictions. For example, he has found that for the SR-52, just plain D.MS will properly round the inexact results produced by y^x, log, etc on integers. This also works for the 58/59. A fix 0 prefix is only required when a fractional part shows up in the display.

Larry has also found that the inexact results of raising 10 to integer powers less than 11 via INV log can be made exact with just D.MS. For the 58/59 this applies to the full range from 0 to 99, since the shorter display mantissa does all the necessary display rounding. All the machines produce exact results with INV log n when n is 20, 30, 40, or 50. For other n between 10 and 100, fix 0 is required to produce exact SR-52 results.

Eigenvalue Calculations (58/59): Bob Thacker (30) discovered that ML-02 could be used to calculate real eigenvalues (in some cases) from real square matrices by the so-called Rutishauser L-R Transformation (iterative) method. The key to Bob's approach is a subroutine call to step 682; but as he notes, in some cases in order to get convergence, pivoting must be suppressed, or the matrix rearranged before being input. Successful convergence reduces the original matrix to upper triangular form, with the eigcnvalues along the diagonal. The following routine may be used with ML-02 to perform a specified number of iterations automatically: Lbl A STO0 **L1'** Pgm 2 C Pgm 2 SBR 682 Dsz 0 1' R/S. 1. Enter matrix elements per ML-02 steps 1-3. 2. Key number of iterations, press RST A; 1 displayed. 3. Examine elements: press Pgm 2 C', R/S, see ith element; repeat for i=1,2,...n^2. 4. Examine row indices: press R/S, repeat n times: If not 1,2,...n, pivoting has occurred. 5. For more iterations, go to step 2; for new problem, go to step 1. Printouts of each successive iteration via the PC-100A would enhance the detection of convergence, and precision growth.

I/O Ideas

An SR-52 air navigation program from Ernst Viehweger (696) provides examples of some handy I/O techniques that may be helpful with other machines, and in other applications involving many I/O parameters, Ernst organizes inputs such that they may be entered in any order and/or combination. Similarly, outputs may be individually specified, with computation time minimized when there has been no change in any of the inputs. Each input is processed via a user-defined key (A-E'), but keys are economized by having the user connect related pairs with + or - operators, later separated by the input routines with 0 =: each input routine ends with either rset or INV stflg. The flag is set by the main processing routine, and tested by a single output routine, determining whether or not a new computation is required. Keyboard integer inputs to the output routine correspond to the register addresses where the desired outputs are located and which are retrieved by indirect addressing.

List/Trace Options Under Program Control And More On Printer Connection Sensing (58/59/PC-100A)

In the course of trying to find ways for a running program to determine printer connection (V2N9p2), A B Winston (707) examined some of the listing options both with and without the printer, when executed under program control. His results lead to the following observations: contrary to the last statement on page VI-4 of the owner's manual, termination of INV List executed under program control does not return control to the keyboard, and both program and label listing can be made under program control without relinquishing control to the keyboard upon completion. The latter are accomplished by having a List or Op 8 instruction at the beginning of a called subroutine which ends with an INVSBR as the last step in the current partition.

For example, if program partitioning ends with step 479, the sequence: ... SBR 476... 476: List stflg 5 rtn executes under program control beginning at SBR by listing steps 477-479, then resumes with the steps following the SBR 476 call. The sequence: *...SBR 475... 475: Op 8 stflg 5 rtn* executes as a label search from step 477 to step 479, and returns to the calling program having effectively done nothing. In both cases, the SBR call without printer connection causes flag 5 to be set. Thus at a cost of only 8 steps, a flag can be automatically set when a program is running without printer connection... better than the 12 steps required by the HIR method (V2N9p2). For this purpose, OP 8 is "cleaner" than List. Then, as A.B. suggests, it only takes 8 steps to do: ...SBR 475... 475: Prt Op 8 R/S rtn which either prints and continues, or halts, depending upon printer connection. R.U. Myers (566) suggests: *...20 Op 7 Op 18 CE...* which in only 7 steps sets flag 7 if the printer is connected, but precludes the use of flag 7 for sensing real errors.

While program call-execution of the List and Op 8 functions may find occasional practical application apart from printer sensing, call-execution of INV List will probably find greater use: It's a cheap way to output tagged results, and do this without a forced halt.

Bob Myers brought to my attention the special trace-control feature of flag 9, which is rather obscurely mentioned on page IV-65 of the owner's manual (instead of on page V-67). Selective trace-printing under program control is a nice feature, but ignorance of this special flag 9 behavior could cause considerable debugging grief!

Tips and Miscellany

No-Hassle Partitioning (59): Roy Chardon (515) suggests recording any/all programs with the 479.59 default partition, then let the program repartition as required, saving the user from having to manually repartition before card read.

Clerical Aids: Mack Maloney (246) suggests writing VxNxpx opposite a membership list entry corresponding to the 52-NOTES applicable correction, obviating the need to squeeze corrections in on the list itself. He also suggests that correspondence to me requiring a reply be written or typed with sufficient left hand margin to give me room to make my reply on a copy. This will save me time, and remind you what I'm replying to. If you don't type, use a soft lead pencil, or black ink, or other writing medium that will work with a thermal copier.

Statistics Keys Tricks: Sandy Greenfarb (200) suggests that clever manipulation of the statistics built-in functions (and/or the statistics Ops) may yield shortcuts to mechanizing unrelated problems (see Dix Fulton's SR-56 challenger program elsewhere in this issue). The idea is to take advantage of the various arithmetic combinations of inputs distributed among the registers used by the statistics functions. I invite the membership to explore the possibilities and to report fruitful results.

Extended Print Code (58/59): Jack Thompson (531) notes that all 100 2-digit integers produce print code. There are no unannounced characters, but even numbered character matrix rows except 8 may be extended 2 characters into the next row by adding columns 8 and 9. For example, 08 prints a 7 and 09 prints an 8 (handy for incremented printout); 28 prints an M, and 29 an N. But 18 is the same as 10, and 19 the same as 11. This print-code behavior is also noted by Carl Seel (328).

More on Joel Pitcairn's Trig Algorithms (V2N9p6): What was incorrectly referred to as a **Byte** magazine error is an approach that can be significantly improved. Joel has since found that setting: Tan $A_0=(x_i+A_iy_i)/(y_i-A_ix_i)$, Arctan $x_0= z_i+x_i/y_i$, Tanh $A_0=(x_i+A_iy_i)/(y_i+A_ix_i)$, Arctanh $x_0=z_i+x_i/y_i$ yields 13 place results with Max=7 (about half the number of iterations originally required). Joel's findings should be of interest to software/firmware designers for full scale computers as well as for the PPCs.

More on Printer Head Cleaning (V2N9p3): As many of you wrote, there are only 100 print elements, not 700, and I stand corrected. However, it seems that there might be more heat buildup in an element that is fired 7 times in the fraction of a second that a line is formed than if it is only fired once. It also appears that if only one character row is printed per line, available heating power is more concentrated. But in any case, the proof should be in the pudding... has anyone actually unclogged a print element using a head cleaning routine with the heavy paper?

CROM Library Notes (58/59): For the Applied Statistics (2) Library, Gerald Donnelly (203) has an improved data entry method for program ST-03. Write him for details. Gene Werner (120) warns that a 3 X 8 inch addendum sheet packed loosely in the box is easy to miss.

PC-100 Modification for TI-58/59 Use: Steve Marum (188) claims to know how to modify a PC-100 printer so it acts like a PC-100A (without battery charging). Write him for details.

SR-52 Mechanizations of Analytic Models: Ron Zussman (88) has written 2 more programs in conjunction with recently published articles: "How to Anticipate Performance of Multipoint Lines" (**Data Communications** July 77 pp51-54) and "Predict System Dependability With a Pocket Calculator" (**Electronic Design** 13 Sep 77 pp100-104). Write Ron at 2456 Ocean Parkway Brooklyn, NY 11235 for details.

Membership Address Changes: 161: 6615 Kentland Ave Canoga Park, CA 91307; 450: 17 Pinewood Dr West Boylston, MA 01583; 538: 5019 Calhoun #232 Houston, TX 77004; 120: 11006 Jean Rd Huntsville, AL 35803; 188: 520 Talley Sherman, TX 75090; 373: 1241 Amherst W Los Angeles, CA 90025; 515: 2527B 25th Loop KAFB, NM 87116; 49: 510 Yeatman St Louis, MO 63119; 365: 60 San Milano Goleta, CA 93017.

More on Card Road Under Program Control (59): A reliable source has revealed a program-execution-architecture feature that renders the parallel processing inference (V2N9p5) incorrect. The observed behavior is due to the way the machine processes instructions: Before a step is executed, the entire contents (8 steps) of the register containing this step must be in a code-execution buffer. When this step is a Write preceded by INV, a prepositioned card is read, then the rest of the code in the code-execution buffer is executed, provided none of it causes a transfer out. Execution then resumes with the first step in the next octet of stored code, which now is new, having just been read in. What appeared to be a bank position dependency was coincidentally a register-position dependency. At most, 7 steps of old code execute following a Write in the first step of an octet.

Ops 20-39 With a 959.0 Partition (58/59): James Merrill (693) found that attempts to increment or decrement Reg 0-9 when they are not within the data partition produce unique flashing displays. Results are different for the 58 with a 479.0 partition than for the 59 with 959.0 I invite explanations for displayed results from the membership.

Tic Tac Toe (59/PC-100A)

Although Tic Tac Toe is more an easy puzzle than a game, its computer/calculator mechanization can be challenging. Fred Fitzgerald (252) leads off with a challenger that follows; press A and follow instructions. Other members are invited to try to out-do Fred with more efficient algorithms, better I/O, etc.

TI-59/PC-100A Program: Tic Tac Toe **Fred Fitzgerald**

Program Listing:

```
000:  Lbl D' RCL*11 Op 02 1 SUM 11 RCL*11 Op 03 1 SUM 11 RCL*11 Op04 Op 05 1
      SUM 11 rtn
026:  Lbl E' RCL 49 Op2 Op3 Op4 Op5 rtn Lbl B' 1 SUM 12 RCL*11 + RCL*9 SUM 11
      RCL*11 +
054:  RCL*9 SUM 11 RCL*11 = STO*12 x=t 5' – RCL 41 = x=t 073 rtn stflg2 RCL11
      STO 19
079:  RCL 09 STO 18 rtn Lbl C' RCL45 SUM1 SUM4 SUM07 0 STO11 D' E' D' E' D'
      RCL45 +/- SUM 01
107:  SUM 04 SUM 07 CLR Op 00 EXC 29 x=t 132 Adv Op 04 CLR EXC 30 Op 03 Op 05
      GTO 428
132:  Adv Adv Adv CLR R/S Prt Adv - 1 = STO10 RCL*10 x=t 160 RCL44 STO29
      RCL 43
155:  STO 30 GTO 111 1 SUM 40 RCL 46 STO*10 RCL 09 INV x=t 187 4 STO 11 RCL
      04 x=t 9' 0 STO 11
183:  stflg1 GTO 9' RCL 58 x:t 20 STO 12 8 STO 11 B' Dsz 11 197 Op 29 B' 1
      STO 11
208:  B' 2 STO 11 B' Op29 B' 2 STO 11 Op29 B' INV ifflg2 237 RCL 19 STO 11
      RCL 18 STO 09
235:  GTO 5' RCL59 x:t RCL28 x≥t 8' RCL27 x≥t 8' CP 0 STO11 RCL0 INV X=t 269
258:  RCL 59 – RCL 23 – RCL 24 = x=t 9' 2 STO 11 RCL 02 INV x=t 289 RCL 59 –
      RCL 23 – RCL 26 =
287:  x=t 9' 6 STO 11 RCL 06 INV x=t 309 RCL 59 – RCL 24 – RCL 21 = x=t 9' 8
      STO 11 RCL 08
314:  x=t 9' Lbl 8' CP INV ifflg1 327 3 STO 10 1 INV SUM 10 RCL 10 ABS STO 11
      RCL*11
338:  x=t 9' GTO 327 Lbl 9' CP 51 STO9 RCL42 STO*11 RCL*12 - RCL58 = x=t 377
      RCL 40 – 4
367:  = INV x=t C' RCL 50 STO 29 GTO C' RCL56 STO30 RCL57 STO29 GTO C' Lbl 5'
      INV stflg2
392:  CP RCL*11 x=t 9' RCL 11 - RCL*9 = INV x≥t 371 STO 11 GTO 5' Lbl A RCL
      39 Op2 RCL 38
419:  Op3 RCL 37 Op4 Op5 E' CP Adv CLR 3 Op17 CMs 6 Op 17 INV stflg1 RCL40
443:  x=t 451 0 STO 40 GTO C' Op0 RCL36 Op3 RCL35 Op4 Op5 RCL34 Op3 RCL33 Op4
      Op5 Adv
474:  GTO C'
```

Prestored Data:

```
33: 3563360000 3341362300 2201200 1731371735 3732170000 37131500
39: 372415 0 360000 320000 4532410015 2317133773 200000002 500000 0 0
49: 2020202020 1635134340 -1 3 -4 2 0 45324100 2732361773 640000
59: 1000000
```

More on HIR Operations (58/59):

R.G. Snow (212) notes that Ops 1-4 copy the display into the 10 LSDs of HIRs 5-8. This is a non-normalized format for data representation (see V1N1p4,5). When such a "number" is displayed, or acted upon by register arithmetic, it is normalized: the digit string is shifted left until the most significant place of the mantissa is non-zero, and the effective multiplications by factors of ten are cancelled by corresponding decrements of the decapower (see V2N1p3,4). In this normalized format, the original print code would not be correctly interpreted, but by adding a 1-digit integer to it, it would be. For xample 1314151617 Op1 Op5 prints ABCDE. 1 HIR 35 normalizes with correct printer interpretation the contents of HIR 5 (Op 5 again produces ABCDE). 1.5 EE ± 7 HIR 55 effectively erases the C, and a subsequent Op 5 produces AB DE. As R.G. points out, since the printer only looks at the mantissa's ten LSDs, multiplications (or divisions) of the normalized contents of HIRs 5-8 by factors of ten do not change printer interpretation. R.G. suggests use of the P/R function to transfer the contents of the X and T registers to HIRs 7 and 8. The sequence: *a x:t b P/R* stores a in HIR 7 and b in HIR 8. He has also found that an SST'd HIR followed by a manually keyed Ind ab performs the HIR operation specified by the contents of (ordinary) Reg ab.

Volume 2 Number 11 48/39 November 1977

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

CROM Use Enhancements (58/59)

New/better uses of CROM code are starting to proliferate, and will be shared in this space as they come to my attention. Priority will be given to Master Library over applications CROMs.

Hyperbolic Trig Functions: Fred Fish (606) has found that ML-05 can be used to calculate the Sinh and Cosh functions: ...52 Pgm 5 C'... for Sinh, and ...STO 2 Pgm 5 E'... for Cosh. If you need Reg 2 for another purpose: ...Pgm 5 SBR 110... and Pgm 5 SBR 006... for Sinh and Cosh respectively with the argument in the display, will also work, and run a bit faster. The user is cautioned that the E' and SBR 006 routines leave the machine in radian mode.

Register Clearing Routine Extension: Fred has also found that ML-01 Routine CLR can be extended to clear registers up through the displayed address by calling it via SBR 012.

Bypassing Stores: Norman Herzberg (668) suggests an improvement to user-program-called CROM programs which devote user defined labels to storing inputs. It is more efficient when accessing a CROM program under user program control to store the input via the program, since the sequence: Pgm ab c takes 3 steps and STO de only 2.

Year Day with ML-20: The following routine converts month, day, and year to year day (0-365 or 0-366): ***Lbl A*** *STO 1 R/S STO 2 R/S STO 3 Pgm 20 SBR 086 - n = R/S* where n=722084 for 1977, 722449 for 1978, etc. To run: Key month (1-12), press A; key day (1-31) press R/S; key year, press R/S; see year day. To calculate n for year y, run this routine for Jan 0, y with n=0 in the routine. For its intended use, tapping ML-20 at step 086 allows separate integer inputs, and speeds processing.

PPC Cryptology

Larry Mayhew (145) suggests that the "unbreakable" trapdoor method (or variations thereof) for coding messages described in **Scientific American** (Aug 77 p 120-124) might be a stimulating challenge for users of the more advanced PPCs. Anyone want to try? I would think that neither the coding nor decoding processes should take more than an hour of program execution time, to be acceptable.

Another approach that has already been explored to some extent by HP-65 users (**65-Notes** V3N1p11) uses a pseudo random number generator (PRNG) as a sort of "one-time pad". For a given message, a specified seed number and the PRNG algorithm are the only required hidden quantities. For encoding, the PRNG is initialized with the specified seed, and run to produce as many outputs as there are characters to

encode. Each output offsets a standard number code for one character. The entire character string is decoded using the same seed and PRNG. The degree of difficulty of breaking the code would appear to depend upon the degree to which different seeds produce different output strings, and whether there is a repeated cycle identifiable in the PRNG output. Members with cryptological expertise are invited to share their comments, ideas, approaches, etc.

Advanced Programming Techniques IV: Topological Sorting

There are important practical uses for the precedence-ordering of certain objects... situations where it is easy enough to determine pair by pair which of 2 elements of a large set needs to be "done" (processed, defined, etc) before the other, but where it is not so easy to order all the elements in one long precedence string such that each required paired relationship is maintained, or to detect the presence of circular relatio nships where an element effectively precedes itself. In Volume I of his **"The Art of Computer Programming"** Professor Donald Knuth introduces a technique he calls topological sorting, which produces an overall precedence ordering consistent with all paired orderings, and which can detect and isolate circular relationships. Computer implementation of this technique requires the establishment and continued update of inter-element relationships as input precedence pairs are examined. After all inputs have been processed, the established relationships are examined to see what order in which to output the elements. A straightforward brute-force (but inefficient) approach might be to assign tables of predecessor elements to each input element. As each precedence pair, denoted here as j.k (meaning that element j precedes element k), is examined, j would be added to k's predecessor table. Elements with no predecessors would be output first. Then successive passes through all the predecessor tables would be made, deleting elements that have been output, outputting elements when their predecessor tables have been emptied. But for large numbers of elements and complex interrelationships, this approach would take a lot of memory, much of which would be wasted in order to assure that each precedence table is sufficiently large to handle all expected cases. Also, such an approach would be slow, due to the requirement for a lot of sequential searching. Fortunately, there are better algorithms, and Knuth devised an elegant, highly efficient one (Algorithm T, page 262), which illustrates the use of both sequential and linked lists, a queue overlayed within a sequential list, and subscripted subscripting. A linked list contains elements which may be scattered throughout a computer's memory, but which are made effectively sequential through socalled link fields: a part of each clement (or the element itself) that points to the next element by containing (or being) its address. For doubly linked lists, each element contains a forward and a backward pointer. One advantage of linking is that insertions and deletions do not require moving large numbers of elements; another, that lists of unpredictable length may be constructed one element at a time from a single memory pool. A queue is a collection of time-related elements, such that the first in is the first out (FIFO), which contrasts with a stack, where the last in is the first out (LIFO).

Algorithm T builds successor lists for each input, drawing from a common memory pool, and establishing forward linking. In a sequential list a count is kept for each input of how many successors it has, the address of each element corresponding to the input identifier.

During processing, as each successor counter goes to zero, it is converted to an output queue link. The program that follows mechanizes Algorithm T and shows how these programming concepts and techniques may be translated into 59ese. Decimal fractions serve as link fields, input identifiers correspond directly to the addresses of a register block (1-30), and indirect and double-indirect (a pointer points to another pointer) addressing handle subscripting and subscripted subscripting. Required memory partitioning is done under program control, with the machine left in a 239.89 partition upon program termination.

Algorithm T may be enhanced with 6 more steps (page 543) to isolate circular relationships (not part of my program), and might be further improved by inputing the number of input pairs, which could then be used to efficiently partition data memory between the sequential list and the linked-list pool.

TI-59/PC-100A Program: Topological Sorting Ed

User Instructions:

1. Initialize: Key n, press E; n printed, 0 displayed.
2. Input Relational Pairs: Key j.k*: 0 < j,k < 31; press R/S; pairs printed, current number of pairs displayed. Repeat for up to 50 pairs.
3. Initiate Processing: Press A; see printed topological sorting of inputs followed by the number of inputs remaining to be output (should be zero). Circular relationships encountered cause output to be aborted, and the number of inputs not yet output will be non-zero.

*Integers j and k identify n distinct objects in a set. Each j.k pair (j≠k) specifies that j is an immediate predecessor of k. In the j.k format, k must occupy 2 places (i.e. for j=5, k=2, key 5.02).

Program Listing:

```
000:  Lbl E x:t 9 Opl7 x:t CMs STO34 39 STO37 31 Op4 RCL34 Op6 Adv 2431 Op4
      CLR
030:  R/S Op6 - Int STO32 = X 100 = STO33 + RCL*37 INV Int = STO*37 1 SUM*33
056:  RCL*32 INV Int + RCL*37 INT = STO*37 R37 ÷ 100 + RCL*32 Int = STO*32 1
      SUM
082:  37 SUM 38 RCL 38 GTO 030 Lbl A Adv Adv 324137 0p04 0 STO38 1 SUM38 CP
109:  RCL*38 Int INV x=t 130 RCL38 + RCL*35 INV Int = STO*35 RCL 38 STO 35
      RCL 34 x:t
133:  RCL 38 INV x=t 105 RCL0 STO36 RCL36 CP x=t 221 0p06 1 INV SUM 34 RCL*36
      INV
158:  Int X 100 = STO37 CP x=t 213 RCL*37 STO31 1 INV SUM*31 RCL*31 Int INV
182:  x=t 199 RCL*37 Int + RCL*35 INV Int = STO*35 Int STO 35 RCL 31 INV Int
      X
204:  100 = STO37 GTO 166 RCL*36 Int STO36 GTO 143 Adv 31351730 Op4 RCL34 0p6
236:  Adv Adv Adv R/S
```

Some New SR-52 Tips and Discoveries

Productive minority member Larry Mayhew (145) continues to explore SR-52 behavior, and notes the following:

1. The 0 ÷ 0 error state is only a special case of the more general case in which any division by zero takes place. Example: 5 ÷ 0 = CE 1 SUM 00 RCL 00 yields -1. All the properties mentioned in V1N1p2, V1N2p2, and V1N7p4 hold. Dividing a register by 0 does not set the special error state, but does cause a flashing display. The following sequence provides a simple way to check for the special error state: 0 PROD 20. If the display flashes, the machine is in the special error state, otherwise not.

2. Conditional tests behave differently with undefined labels than they do with undefined absolute addresses. The sequence: ifzro C (where C is undefined) will cause a flashing display regardless of whether the display is zero. However, the sequence: ifzro 999 will cause a flashing display only when the display is zero. (A special case of this has been noted in V1N4p6 for testing flags.) Thus, if one can use an error condition as a flag (V2N1p4), the sequence: ifzro 999 sets a flag (error condition) only when the display is zero.
3. I have long had the impression that the only difference between absolute and relative addressing is the amount of space and time they take. However, there are special cases where absolute addressing can be used to much advantage over relative addressing. For example, beginning at 000 write: Lbl A ifzro 008 R01 SUM00 HLT. This has the effect of incrementing register 00 by 1 if the display is 0, but by the amount in Reg 1 if the display is non-zero. A few days ago I found a practical application for having an address branch to itself: Beginning at step 099 write: ifzro 100 STO 00, which puts any non-zero value into Reg 00, but which puts 100 into register 00 if the display is zero. Such tricks save space only under special conditions, since they depend upon being able to structure one's program just so.
4. If in doubt about using p73 (V2N7p4) one can write simply: RCL p73, and the transfer always takes place without an error being caused.
5. One should note that the short absolute branch mentioned in V1N7p3 will work with unconditional as well as conditional transfers. For example: GTO STO 0 causes a transfer to 000 (works with SBR as well). Furthermore, it can be very useful to put the short conditional branch at the beginning of program memory, as this has the effect of simply skipping a certain number of steps if the test is met. For example: at step 000 write: Lbl A ifpos STO9, and steps 5..8 are skipped when the display is positive (though of course an error condition is set).

Label Rules (58/59)

The SR-52 LBL LBL tricks won't work for the 58 or 59. Apparently the Lbl code (76) following a conditional test instruction is not recognized as the Lbl instruction during a label search.

Rusty Wright (581) notes an inconsistency in the owner's manual concerning the use of Lbl as a label. Page IV-43 says it cannot be used and page V-56 implies that it can. It appears that Lbl Lbl will work only for the unconditional transfers: GTO Lbl and SBR Lbl, and a further restriction is illustrated by the following sequence: GTO Lbl B Lbl Lbl R/S. Execution starting at the GTO does as expected: normal transfer to the R/S occurs. But a call to B behaves as though B were undefined.

As the manual says: 2nd, LRN, Ins, Del, SST, BST, and Ind are not valid labels; p82 (HIR) is the only pseudo that is. Although the manual cautions the user not to use R/S as a label, the reason given is not the primary one. As Jared Weinberger (221) discovered, a subroutine labeled R/S placed near the top of memory is sometimes uncallable by SBR R/S ...execution can be an effective GTO R/S. There appears to be a subtle machine-state dependence affected sometimes by CLRs and RSTs. Further investigation of this phenomenon is invited.

On a conditional transfer to a labeled address, the 58/59 (unlike the 52) make a label search only if the condition is met. This has 2 consequences: 1) execution is faster for an unmet condition, particularly when the labeled address is far along in a program, and 2) if the label is undefined, an error state is produced only if the condition is met. This provides a handy way to produce a conditional halt with flashing display.

Extending Data Memory via Efficient Packing

Many PPC users have at one time or another saved storage space by putting 2 numbers in a single register using the mantissa decimal point as a convenient separator; and for one and 2-digit positive integers, it is easy to store 5 to 13 quantities per register by decimal place separation. While these 2 common methods work, they are often wasteful. An optimum approach should just comfortably handle all expected data values, and minimize the execution time required to pack and unpack each datum. Before deciding on a particular approach, one should first determine the max and min data values that can be expected. A scaling offset may be added to each datum to make all data positive, and a multiplying factor applied to eliminate fractions. The difference between the scaled max and min values determines how many data will fit into one register. For data which are so-called Boolean variables (they assume one of only 2 values or states), use can be made of the mantissa and decapower signs. Thus at one extreme, you could squeeze 45 (43 in the mantissa, and 1 for each sign) Boolean variables into one register, and at the other, only one, where data values span a range or more than a million. Between these 2 extremes, it would be wasteful to have to reserve 2 decimal places for each datum that falls into a scaled range of say 0-11, the limit being 6 data per register... the same as for a 0-99 range. But if use is made of another radix (twelve is best for this example), the limit is doubled (see an analogous radix sixteen application in V1N5p4). Following along the lines of routines A and B in V1N5p4 we can (in 58/59ese) pack and unpack up to 12 numbers in the 0-11 range in Reg 1 with:

```
Lbl A + 12 Prd 01 0 = SUM 1 rtn Lbl B  RCL 1 ÷ 12 - Int STO 1 = X 12 = fix 0
EE INV EE rtn.
```

Note that since the newer machines do 13-place display arithmetic, there is no point in mechanizing routine C. The manner in which packed data need to be addressed will impact the complexity of addressing mechanisms. Suppose using a TI-59 in the above example, we wish to pack up to 1080 numbers in the 0-11 range in Reg 0-89. If we need only to pack sequentially and unpack in reverse order, also sequentially only once, with no partially filled registers, the following routines would do:

```
Lbl A 12 STO 98 Lbl 1' R/S + 12 Prd*99 0 = SUM*99 Dsz "98" 1' 1 SUM 99 GTO A
Lbl B 12 STO 98 Lbl 2' RCL*99 ÷ 12 - Int STO*99 = X 12 = fix 0 EE INV EE R/S
Dsz "98" 2' 1 INV SUM 99 GTO B
```

where CMs A initializes routine A, routine B is initialized by A, and the routines are run by: A, R/S, R/S,... and B, R/R, R/S,... . But if we need to be able to pack or unpack randomly and more than once, things get more complicated. Members are invited to address this problem, and to share their best mechanizations. Of particular practical use would be the efficient storage of coded personnel information, or business transactions. In some cases it will be advantageous to group data by ranges to make packing more efficient.

Post-printing exercise of the above routines revealed the following qualifiers: there are specific sequences of 12 numbers in the 0-11 range that can cause packing overflow due to the non-rounding of the 13th place, and following the use of the latter routine A, Reg 99 needs to be decremented by 2 for proper routine B initialization. In general, it's best not to use more than 12 decimal places for packing. The formula for determining how many digits (n) in radix r can be packed into a d-place decimal number is: *n=Int(d÷logr)* and thus for d=12, the max number of radix twelve digits should be 11, and for radix two (Boolean): 39 plus the 2 that the signs can represent.

The consequences of machine inexactness are many, and often insidious, as many have found from painful experience, but since inexactness is a basic characteristic of digital machines, we might as well learn to live with it. Oscar Louik (745) reports a related problem: In using the decimal point separation method to get one TI-59 register to hold 2 3-character Op4 Op6 prompting words, he found that 10 y^x 6 = as a multiplier did not properly convert a six place fraction to an integer. 6 INV log is just as bad, and the problem results from the fact that both sequences produce 999999.9999959, not 1000000, and that Op4 only acts upon the integer part of the display. The sequence: ...EE 6 INV EE ... will multiply a displayed fraction by exactly a million, and is a better way to unpack the fractional part. As Oscar discovered, in this application it is best to make the most commonly used prompting words into the integer parts, since no multiplier is required, and Op4 even does its own integer extraction.

Routines (58/59)

Unit Step and Delta Functions: Roy Chardon (515) notes that the signum function (Op10) can be used to evaluate the Heaviside Unit Step (H(x)=0 if x < 0, H(x)=1 if x ≥ 0), and Dirac Delta (delta(x)=0 if x≠0, delta(x)=1 if x=0) functions ... for H(x): *(Op10 + 1) Op 10*; and for delta(x): *(Op 10 ABS - 1) ABS*.

Angle Mode Indicator: Jared Weinberger (221) found another use for Op10: Lbl A 90 cos Op10 rtn returns 0 when in degree mode, -1 for radian, and 1 for grad.

Digit Reversing (56,57,58,59): Jared also has a TI-59 digit reversing routine that handles ten-digit positive or negative integers: ***Lbl A*** *(CE ÷ 10 Prd 1 - Int STO 2) SUM 1 RCL 2 INV x:t A (Exc 1 X 10) rtn,* where for the 56 and 57, Lbl A is omitted, and the A following the x=t is replaced with the address of the first step, or for the 57, A is replaced by an available label.

SR-56/TI-57 Program Exchange

Dave Johnston (5) has a new catalog listing 101 SR-56 programs, 6 of which have been translated for TI-57 use. Dave continues to provide his exchange service at bare cost (5¢ per page), but even with a little help from the Club I'm sure he appreciates extra contributions. Dave's also looking for more inputs to his library: 56, 57, and 58 programs.

Membership Address Changes

219: Deceased; 671: c/o ARAMCO Box 9999 Dhahran, Saudi Arabia; 149: 1205 Akers Las Cruces, NM 68001; 697: 21 Kristin Dr #718 Schaumburg, IL 60195.

Volume 2 Number 12 48/39 December 1977

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

Routines

Double Precision (56,57,58,59): Norman Herzberg (688) suggests that putting together a library of optimized fully double precision arithmetic routines would be a worthy endeavor for the membership. He leads off with the following integer multiplication routine which runs fast and appears to give correct results to 20 places. Printing the product on a 20-digit line might be a useful enhancement. To use Norman's routine: Key 10-digit multiplicand, press A; key 10-digit multiplier, press B, 10 MSDs of the product displayed; press x:t, see 10 LSDs of product. Multiplicand remains intact for use with a new multiplier.

```
Lbl E' ÷ x:t 1 EE 5 = Int x:t - x:t X x:t 1 EE 5 = rtn Lbl B E' STO4 x:t INV
SUM4 X RCL 1 STO 7 = STO 5 x:t X RCL2 INV SUM7 = STO6 E' Exc6 + x:t + RCL7 X
RCL4 + RCL5 = E' X 1 EE 5 = SUM6 x:t SUM5 CLR RCL6 x:t RCL5 rtn Lbl A STO1 E'
STO2 x:t Exc1 rtn.
```

Members are invited to try to improve upon Norman's routine and/or address other double precision functions.

The Mathematical Integer Function (56,57,58,59): Joel Pitcairn (514) notes that the built-in Int function does not handle negative reals properly, since for mathematicians the integer part of x is defined as being the greatest integer ≤ x. Joel suggests using the following routine (slightly revised) to get the integer part of all reals: *...CP + INV Int Op 10 x≥t 1' + L1' 0 = Int...* . I invite 56/57 versions (which will have to do without the Op 10 signum function).

More Conditionals (56,57,58,59): Although data can usually be arranged so that the conditionals x=t, x≥t, and their inverses will handle all required comparisons, there are times when it would be handy to have x≤t and its inverse as well. Jared Weinberger (221) suggests: ...x:t x≥t... for an effective x≤t, and ...x:t INV x≥t... for an effective x>t. Of course, the user needs to take into account the x-t reversal in subsequent processing.

Automatic Number Printer (58,59) Revisited (V2N10p2): It has been brought to my attention that R G Snow's routine might have a flaw, since an input 88888 returns 1111111109 instead of 1111111111. Well, it turns out that RG's routine translates 7s or 8s at either the MSD or LSD position into 8s or 9s, respectively, instead of 11s or 12s. But this is all right since print code 8 prints a 7, and a 9 prints an 8 (see V2N10p4).

Membership Address Changes

203: 9423 Bickford Rd Kernersville, NC 27284; 533: Mink Creek Rd Star Rte Box 431 Pocatello, Idaho 83201.

---

interested person: $6.00 includes six future issues of 52-NOTES; back issues start June 1976 @ $1.00 each.

Tips and Miscellany

TI-58 Users Assistance: Roy Chardon (515) has a TI-59, and has kindly volunteered to broaden my offer to 58 users (V2N7p5): On a *"to an extent determined by the circumstances"* basis, Roy will 1) attempt to "complete" problems which won't fit on a 58, and 2) will serve as a PPX-59 channel... both "*on a best effort basis, with no money (or mag cards) exchanged.*" Note that Roy's address changed recently (V2N10p5).

A p31 Application (52,58,59): Roy notes that p31 can be put to good use in programs requiring the user to write special applications code. For example, replace the R/S at step 373 of the V2N8p5 program with p31, and delete the first LRN in step 1 of the user instructions.

References to the 58/59 Owners Manual: I plan to continue refering to the original (dash one) version. When 52-NOTES references identify errors or omissions corrected by later versions, there is no harm done, but in cases where apparent contradiction or confusion results, I ask that cognizant members so inform me.

CROM vs User Code Execution Speed (58,59): Tom Ferguson (421) found that CROM code executed within the module runs faster than the same code downloaded into user memory and then run. For example, the routine C part of getting 69! with ML-16 takes almost twice as long to execute in user memory as it does in the CROM. Members are invited to explore this further, and see if there are certain functions or sequences that show more marked CROM/user-execution speed differences than others.

LRN Mode Lockout (58,59): Fred Fish (606) notes that manual partitioning that leaves the program pointer out of bounds disables the LRN key. An RST restores LRN mode access, although a 0.59 TI-58 partition absolutely disables the LRN key as long as that partition is in effect.

More on Dsz (58,59): Jack Thompson (531) notes an exception to all-register Dsz-ing (V2N8p2): Dsz "40" is interpreted as Dsz Ind. So don't try to Dsz Reg 40 directly.

Ideas for a "Support Software" CROM (58,59): Several members have suggested that a CROM composed of widely used program building blocks would be especially useful. Send me your candidate routines or ideas, and I'll consolidate them and see if we can interest TI.

More on Printer Head Cleaning: John Allen (104) reports that printer head cleaning with the heavy paper really works (V2N10p5). His PC-100A failed to print certain columns when he first bought it, but performing the cleaning instructions fixed it right up.

Protected Editing (58,59): If you don't want to disturb unrelocatable code near the bottom of program memory while editing code near the top, temporarily repartition to make into data the code to be protected.

INV Viability (58,59): The lack of INV viability noted in V2N6p3 does not apply to INV instructions placed before labels. For example: Lbl A INV Lbl B Sin rtn executes as sin when called by B; as INV sin when called by A. Also, the normally merged rtn (INVSBR) can be broken up as: ***Lbl A*** *INV Lbl B SBR 1' R/S* ***Lbl 1'*** *...* which if called by A returns without doing anything; calls SBR 1' with a call to B.

Mag Card Printing (52,59): Horizons Technology Inc is marketing ways to letter mag cards. For details, write Tom Schroeder (663).

SR-51/51A/PC-100A: Several members have called attention to the **Byte** (Sept 77 p 176) letter describing SR-51/PC-100A compatibility. Michael DeTraglia (544) found that his SR-51A doesn't have to be "jammed" on to work. For best results, select SR-52 switch position, and trace mode. Printout of non-SR-52 mnemonics suggests that the extra SR-51 functions might at one time have been considered as part of the SR-52 design. Leon Ablon (619) notes that the PC-100 also works in this SR-51/51A application, and that for dollar totals the 51s are better than the 52 since all addends are printed to 2 decimal places with a fix 2 format (the 52 drops trailing zeros, as do the 55, 58 and 59).

SR-52 Extended Memory: Robin Wooding (350) has for several months been using an SR-52 modified with extra memory such that 160 of the normal 224 program steps can be manually switched out and replaced with 160 new steps, or vice versa. Write him for details of some of his applications.

HLT-R/S, rset-RST, and rtn-INVSBR 52-58/59 Differences: While the 52's HLT hardens a soft display, the 58/59's R/S does not. Bob Myers (566) notes that a programmed 52 HLT with one or more pending subroutine returns followed by a manually keyed A-E' that ends in rtn causes control to move back through the original calling path. For the 58/59, following a program-executed R/S and the manual A-E' the next INVSBR encountered executes as an R/S, and effectively clears the subroutine return address register. Upon encountering an rset, the 52 clears a soft display, while the 58/59 RST does not affect the display.

Fractured Display (58/59): Rusty Wright (581) found that in LRN mode at the last step of the current partition, keying a multi-step instruction causes the display to be reformatted in strange ways. For example, with a 59 at turn-on, key 1234 GTO 479 LRG GTO SBR, and see ".00000 1234". The number of digits in the preset display appears to determine the number of zeros appearing to the right of the decimal point. Other rules prevail with a floating point (scientific) display, and Dsz a b or Dsz*cd e at step 479 do even stranger things (the contents of reg cd shows up in the display).

Corrections: Gerald Donnelly (203) and Roy Chardon (515) note that I got the "mental" and "spiritual" categories interchanged in translating from Heinrich's German (V2N9p3). Interchanging the register pairs 19-20 with 21-22 should set things straight. And, as printed, Bob Myers' printer sensing routine (V2N10p4) sets flag 7 when the printer is not connected.

Decapower Zero Suppression (52): Douglass Darrow (687) notes that the SSTd return from a called subroutine suppresses decapower zeros just as the unmet conditional test does (V2N7p4).

More on INV' and Dummy operations (56,58,59): Rusty Wright (581) notes that for the 56, INV' supplies a missing operand, but requires 2 successive = or) strokes for completion of the pending arithmetic. The Owners Manual specifies only CE as a shortcut to supplying missing operands, but Rusty finds that RST also works. The 58 and 59 act like the 56, except that STO, RCL, and SUM also produce missing operands when keyed manually, but as Jared Weinberger (221) found, when executed under program control treat the intended operation as a register address.

Program/Routine Listings for TI-56,57,58,59 Use: 52-NOTES articles whose titles are followed by: (56,57,58,59) or some subset thereof are likely to contain listings written for TI-59 use. Generally no changes are necessary for the 58, although in some cases, register addresses and partitioning need to be scaled down. For the 56, labels and their associated relative addressing must be replaced with absolute addressing, and for the 57, available labels should be substituted for specified A-E' ones. For both the 56 and 57, register assignments should be carefully assessed vis-a-vis existence and special use (such as by statistics functions). By and large, if a PPC's identifier doesn't appear in the title, application of associated routines may require non-trivial translation effort (see V2N8p6).

SR-52A Zero Divide: Paul Blair (526) notes that the special zero divide error state (V2N11p3) does not occur with his SR-52A (V2N5p3).

Flashing Display Variations: Mark Stevans (216) notes that there are 2 types of flashing display for the SR-52: 1) Display is com- pletely blank between flashed positive numbers > 1, and 2) The 2 minus signs appear dimly between flashed numbers. Mark refers to the first as a hard flash, and the second as a soft flash. Dividing by zero or taking the INV sin of a number > 1 produces a hard flash; keying an undefined label produces a soft one. An SR-56 error condition caused by a manual or SSTd STO INV' (discovered by Rusty Wright) is soft. I haven't yet found a way to produce a soft flash on the 58 or 59.

More on Execution of Unintended CROM Code (58,59): Rusty Wright found a code 81 at step 139 of Pgm 3 of the Statistics CROM, and reports that it executes as RST: control returns to step 000 of user memory (see V2N8p6). Rusty also found a couple of code 31s (steps 074 and 077) in the same CROM program, and found that they are ignored ... like p31 in protected code (V2N9p4).

Another SR-52 Analytic Model (V2N10p5): Ron has another program, titled: Computer Queuing Analysis on a Handheld Calculator (COMPUTER DESIGN Nov 77 pp85-94). Write Ron for details.

Quaternion Arithmetic (59): Joel Rice (3) has written programs to perform quaternion (spinor) arithmetic for both real and complex inputs, and is prepared to share his listings with members who send him a SASE with 2 stamps.

Friendly Competition

At least one HP-25 user has conceded that 100 unmerged SR-56 steps amount to more effective memory than 49 merged HP-25 steps (comment by Jim Davidson **65-Notes** V4N8p21)... and Jim has done a lot of HP-25 programming. Both machines are still powerful PPCs for the money, but the SR-56 has always cost less, and so far, no HP-25 program has come to my attention that couldn't be duplicated or bettered on the SR-56.

A year ago in a **65-Notes** article on Taylor Series Summations on Programmable Calculators (V3N9p17), Jim challenged users of AOS machines with: can you *"...do as well as RPN for Taylor's series? Or even come close?"* citing some HP-25 routines he had written. Well, Reinhold Patzer (689) has risen to the challenge with some SR-56 routines, all of which take less effective memory than the corresponding HP-25 routines, and produce results more accurate by a factor of 100. As a specific SR-56 response to Jim's benchmark program for e^x (**65-Notes V3N9p42 - Program 3),** Reinhold offers:

00: STO0 1 STO1 X x:t RCL0 ÷ RCL1 X (CE + x:t X x:t 1 SUM 1) INV x=t 06 **=** *x:t R/S,* which in 30 steps takes less effective program memory than Jim's HP-25 routine*: 01: STO0 0 STO1 1 STO2 RCL2 RCL1 1 + STO1 ÷ RCL0 X STO2 + x≠y GTO 06 GTO 00,* where 18 HP-25 steps use 36.7% of memory compared with Reinhold's 30% (29% if one omits the last x:t, which appears to be unnecessary). On a test case of e^{25} I find that Jim's routine gets 6-place accuracy in 64 iterations, taking 46.1 seconds. Reinhold's gets 9 places in 71 iterations, taking 67.3 seconds. Thus it would appear that Jim's runs significantly faster, at 1.39 iterations/second vs Reinhold's as 1.05, but it would take Reinhold's only 50.4 seconds to get 6-place accuracy. In his article, Jim points out the HP RPN features that facilitate his approach, but it is worth noting that one of the key features: datum replication from T to Z when the stack is dropped is nicely matched in this exercise by the SR-56 T register feature that its contents is not altered by AOS stack movement, or arithmetic operations. It should also be noted that AOS machines without the T register (notably the SR-52) would be at a significant disadvantage in this competition. Another point: When HP-25/SR-56 routines are being compared, if their application is enhanced by being subroutine-callable, as might well be the case in this exercise, the HP-25 loses. However, if Reinhold's routine is called by a main program, there is a pending arithmetic operation that must be cleared following disposition of the results if further use of the arithmetic stack is to be made.

Challenges aside, I recommend Jim's article to PPC users who need help in formulating problems and synthesizing algorithms. Jim takes the reader from brute force, inefficient, but easy-to-understand approaches to trickier more efficient ones, which as Reinhold shows can apply to AOS as well as to RPN machines. Reinhold acknowledges that Jim gets the credit for tricky exiting, and efficient x^i and x!

Hal Brown has succeeded in getting a new version (**65-Notes** V4N8p10) of his 5 X 5 matrix program (52-NOTES V2N8p3 and V2N10p2) to solve more special-case problems without manual intervention, and appears to have eliminated the error-without-warning cases. Interested members may send a SASE to Richard Nelson (2) for copies of this latest version, but are advised to insert 2 missing steps: 50: GTO 0 and 150: x≠0. I invite comments. The only other HP-67/97 5 X 5 program to come to my attention is in the "High Level Math" volume of HP's recently published Users Library Solutions. This is a 1-card program by John L Gustafson Ruddock House Caltech Pasadena, CA 91126 that calculates the determinant and inverse by a method similar to Barbara Osofsky's (V2N5p5), requiring no manual permutation of rows or columns, but which is less compact, slower (takes about half an hour to run), and does not handle ill-conditioned matrices as well as either Barbara's or Hal's. For the ill-conditioned matrix A whose elements are $a_{ij}=1/(i+j-1)$, John's program doesn't quite get one-place accuracy, Barbara's gets 4, but Hal's wins with 5 (TI-58/59 ML-02 gets 8). Here is a situation where for a particular approach the greater arithmetic precision of the TI machines is significant. If Barbara's program were to be run with 10-digit precision, results would probably be similar to John's. Hal's and ML-02 both employ pivoting, and I would expect ML-02 results to be similar to Hal's if produced with 10-digit precision.

Keying Printer Character Code (58/59)

While the Rausch Overlay method (V1N3p2) works well to number-code the alphabet for non-alphanumeric machines, it does not handle all 64 of the 58/59 print characters, and requires code-translation-machine-execution time. Lou Cargile (625) has tried a different approach, which in one form or another looks promising: He has typed/written out the 64 characters on strips of paper which he fastens conveniently on the printer. Characters are grouped according to the first digit of their print codes, and are annotated with the second digit. For example, associated with the numeral 2 key are the characters FGHIJKL which have as their respective second digits 1234567. Some might find it easier just to string out all 64 characters with the corresponding 2-digit print codes beneath each; or perhaps Lou's method could be sufficiently miniaturized to make feasible a direct keyboard overlay. Whatever the approach, the goal is to optimize the speed and accuracy with which the user converts the characters to 2-digit keystrokes, until such time as he has memorized them. Other ideas are invited from the membership.

Editorial: 52-NOTES Style/Club Philosophy Revisited

A majority of members who comment on 52-NOTES style and content continue to like things as they are. Complaints haven't changed much since we began, and generally reduce to: 1) material is too technical and/or terse, 2) important items are buried in long paragraphs, 3) abbreviations/mnemonics are not defined often enough, and 4) membership expiration notices should be issued. The nature of these complaints suggests that the few dissatisfied members regard their membership as a one-way magazine subscription, and I recommend that they reexamine the Club's purpose (V1N1p1), and my approach to writing 52-NOTES (V1N5p1, V2N7p1). People teaching technical disciplines often find it helpful to point out to their students that they should not consider such courses to be spectator sports. Learning to use modern computing machinery is no exception, and I reemphasize the importance of actually working examples on your PPC, and carefully and thoroughly covering all references. Don't skip over something you don't understand at first glance; dig into it, and write for help if need be. One advantage of a 6-page newsletter is that it is not impractical to re-read many back issues from time to time... a good way to put lost or overlooked tools within reach, and to refocus your attention on topics whose value to you has waxed since first acquaintance.

I'm pleased that the ranks of the productive minority are growing, but suspect that there are still many in the silent majority who are hiding their lamps under the proverbial bushel. Members who participate the most seem to have the least difficulty getting all the information they want from each issue of 52-NOTES, and they rarely miss an issue due to forgetting membership expiration dates! I think it benefits all concerned to keep 52-Notes' technical level reasonably high, otherwise the top contributors will lose interest.

As the end of 1977 approaches, I take this opportunity to wish you and yours a happy holiday season, and another challenging year ahead.

Volume 3 Number 1 48/39 January 1978

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

Fractured Digits (58/59/PC-100A)

Jared Weinberger (221) is the first (and so far only) 58/59 user to bring to my attention a means of creating the SR-52's fractured digits: ° " ' - and blank (see V1N2p5). Jared's approach is an extension of Rusty Wright's procedure for creating a fractured display (V2N12p3), but requires printer connection. Jared found that the sequence: *1234 GTO 479 LRN GTO TRACE* ÷ produced the ° and – symbols in the display of his TI-59 at turn-on. Further investigation reveals the following: 1) Only 2 fractured digits are produced, and they are always in the format: .nnnnfnbnnnf where n=numeral, f=fractured digit, and b=blank; 2) The f on the right is a - or a ° depending upon whether the display is fixed or "full" floating (decapower MSD≠0); 3) The f on the left is determined by the current partition; and 4) Somewhat different effects are caused depending upon what keys are substituted for the second GTO and the ÷ above, and the number and formatting of displayed digits. For alternate partitionings, the first GTO above should be followed by the address of the current last step, and for both the 58 and 59, the following hold: 479.XX produces a °; 3 99.XX an ', 319.XX a blank, and 239.XX a "; 559.XX with the 59 produces a -; other possible partitions produce redundant results or show a numeral in the left f position.

Jared's discovery confirms what many of us had suspected: that if a way could be found to produce them, the 58/59 fractured digits would be the same as the 52's. What seems strange is the dependency upon printer connection (you still can't print them), partitioning, and why they appear in only 2 display positions, and where they do. Other members are invited to explore further, and as Jared suggests, maybe we will eventually be able to control fractured digit generation and positioning to the extent now known to be possible with the SR-52. Incidently, Jared notes that trying to make the 58/59 believe it is connected to the printer by setting flag 9 won't work!

Designing a Program/Data Comparator (59)

Software developers sometimes use code-comparing utility programs as debugging tools when the differences between 2 versions of a program or routine need to be detected and isolated, or when there is reason to believe that the same program or routine stored in 2 separate memory devices is not represented by the same binary configurations.

The same also applies to PPC program development, and the TI-59 with its card reader and large partitionable memory ought to be able to make fast, accurate code comparisons to help the user troubleshoot a variety of practical problems.

Data comparisons are easy enough to mechanize: pointers can be moved through 2 banks, accessing paired registers for x=t comparisons, with mismatches appropriately flagged. But program code can only be compared when it is treated as data, and here the code transfer rules complicate things. Not only do the SR-52 rules apply (V1N2p5), but also if an 8 or 9 is at position B, the overflow number 9.99... D99 is created upon transfer, regardless of what the other 7 step-codes are. So here is a challenge: Write a 59 or 59/PC-100A program that will compare the 240 steps of one cardside with those of another, detecting all possible mismatches, and minimizing false alarms and user intervention. It might be advantageous to make use of the program-controlled List capability (V2N10p3).

Routines

Double Precision Multiplication (56,57,58,59): Taking advantage of guard digits operation, Mike Louder (53) has written a routine about half the length of Norman's (V2N12p1). I've taken the liberty of paring Mike's even further, resulting in having the user bisect each factor with a decimal point for the routine listed below. The key to Mike's approach is forming the 10 MSDs of the product by Int(F1 X F2) where F1 and F2 are the complete input factors. This assures that any carries from the MSD of the fractional products occur properly. Similarly, carries from lower fraction products are properly handled by summing only the fractional parts of each product.

```
Lbl A STO1 X R/S STO2 = Int R/S RCL1 INV Int X x:t RCL2 Int + RCL1 INT B +
x:t Llb B X RCL2 INV Int = INV Int rtn.
```

To run: Key multiplicand, press A; key multiplier, press R/S, see 10 MSDs of product; press R/S, see 10 LSDs of product preceded by a decimal point. Input must be in the form: ddddd.ddddd .

Efficient Polynomial Evaluator (58/59): Fred Fish (606) has a routine of the form: ***Lbl A*** *RC*ab = X RCL cd + Dsz "ab" A RCL 0 = rtn* which neatly mechanizes the nested multiplication polynomial evaluation method without using parentheses. Registers ab and cd should be at the end of available data memory, with Reg ab initialized with n (the order of the polynomial) and Reg cd containing the input value for the variable x. Registers 0, 1, ... are initialized with the coefficients for x^0, x^1, ...x^n, within the limit n < ab. For new case, reinit Rab & Rcd

Regulated Clock (58/59): Dix Fulton (83) has devised a routine that combines a timer with the means to automatically correct it with:

```
000:  RCL0 + INV D.MS Pause x:t RST Lbl A Fix9 ((D.MS - RCL1)/(x:t D.MS +
RCL0 - RCL1)) Prd0 x:t CLR x:t R/S Lbl E D.MS Fix4 STO1 + RCL0 + RST.
```

To run:

1. Enter approximate time correction factor (try .0004), key STO 00
2. Start clock with current time in display: hh.mmss, press E.
3. To regulate: Press R/S during a pause, after clock has been running awhile; key exact time of pressing the R/S, and press A; see delta correction factor displayed; go to step 2.

Fractured Digits Synthesizer (52): Dick Blayney (610) has a nice short routine (slightly revised as listed here) for creating fractured digits in all but positions 0, 10, and 11 (per V1N2p2 numbering), which substitutes arbitrary numerals where no fracturing is specified, producing somewhat the same results as the longer V2N3p4 one:

```
000: 1/x 1 3' 7' yˣ lnx 1 E' Lbl A X 1 EE +/- 12 + 1 = STO99 INV EE HLT INV
log EE fix0 EE +/- PROD99 INV fix RCL99 ÷ SUM60 RCL70 rtn.
```

To run:

1. key 10 digits to produce characters at positions 1-10 per the masking rules (V1N2p5), press A;
2. Key 2 digits for the position 12 and 13 characters, press RUN.
3. Press =, and see the synthesized fractured digits display. For new display, press CLR and go to step 1.

Last Digits Viewer (52): Dick has a variation on Jared's V1N4p6 routine, which suggests that there are better sequences than the ... STO + 60 ... that we've been using for display fracturing. Dick's *216: **Lbl E** ÷ SUM60 = p31* is the same length as Jared's, but doesn't cause wipeout with negative numbers, and appears to properly show the 10th through 13th digits of any machine number. The X operator appears to work as well as ÷ in this routine, but substitution of PROD for SUM incurs the wipeout problem Further investigation of various sequences of the form: ...A B 60 = ... where a is a dyadic operator and b is a register operator may prove to be fruitful.

A Short Closed-Form Fibonacci Number Generator (56,57,58,59): Taking advantage of the fact that the second term in the closed-form formula (V1N4p6) is always small enough to be neglected if results are rounded to the nearest integer, Joel Pitcairn (514) suggests: *x:t* y^x *x:t ÷ 4* $\sqrt{x}$ *+ .5 = Int,* where the T register in initialized with (1 + $\sqrt{5}$)÷2, and the sequence executed with n in the display. Note that the constant stored in the T register is preserved indefinitely... an advantage applicable to other problems requiring repeated use of a constant.

Error State Reversal (52): Following Jared's path (V2N10p2), John Allen (104) points out that ...INV iferr LBL LBL iferr CE... will alternately cause and clear the error indication. This takes one less step than Jared's and saves another step in sensing the condition. Note that CE can't appear as a label for this to work.

Tips and Miscellany

Friendly Competition: Mort Schwartz (199) has responded to the challenge of Karl Hoppe's 70 digit square program (V2N5p5,6) with one written for the HP-67/97. I have tried a number of examples, and they all appear to work. However, Mort's program takes more than twice as long to run. Interested members may write Mort or me for copies.

Strange CROM behavior (58/59): Roger gentry (398) decided to try out sequences of the form Pgm mm R/S (suggested on page V-62 of the Owner's Manual, but acknowledged by TI as a mistake). Results appear to be program-dependent as well as machine-state-dependent. Roger also found strange results from related sequences of the form: Pgm mm SBR N, where N is undefined in Pgm mm. Members are invited to report results. While such efforts may not appear to lead to practical applications, there is the possibility that important aspects of CROM execution will surface.

Magnetic Card Extraction (59): Joel Pitcairn (514) suggests restarting the drive motor with another card when a just-read card is hard to remove. Stop the motor with R/S.

Custom CROMs (58/59): TI is currently accepting orders for custom designed CROMs. For $25,000 is will fabricate up to 1000 copies of up to 99 customer written programs. Call Dick Cuthbert at TI's Lubbock facility: (806) 747-3737 X470, for details.

Printer "Hiccup" (58/59/PC-100A): Jared Weinberger (221) has found that trace execution of an Ind operation on a register outside the current partition causes the printer to stop momentarily and print a bit of nonsense in place of the out-of-bounds address.

Soft Flash (58/59): Rusty Wright (581) has found that attempted INV sin execution of a number less than -1 produces an error display similar to Mark Stevans' definition of a soft flash (V2N12p4): the faint C flashes along with the displayed number.

A University Association of Calculator Programmers: Bob Moore (488) has been organizing students and faculty at the University of Washington who, among other things, have participated in "...calculator math labs on campus and manufacturer-sponsored programming seminars in the Seattle area." As a fund-raiser, Bob's group is marketing the "...new uncompromising Zlotnian Calculator." Interested members may write Bob for details. Bob is one of the few educators to come to my attention who is examining some of the important pros and cons of calculators and PPCs in the classroom, and has already addresses some of the important considerations that all educators must sooner or later confront. Other members with links to the academic community may find it enlightening to discuss this topic with Bob, and/or share their views via 52-NOTES.

Successful Mag Card Recording Check (59): When at least one spare memory bank is available, Bill Adler (599) suggests using it to verify newly recorder code/data from another bank before losing it at machine turn-off, as there are instances where cards appear to record correctly, but are later found to read improperly.

Agriculture Programs: J.E. Dunn (791) reports the availability of a collection of SR-52 and TI-59 programs sponsored by the Iowa State University Cooperative Extension Service to aid farmers. Write J.E. for details.

Interrupt Processing (58/59): Although the new machines don't have the handy manual angle-mode switch for interrupt processing (V1N2p3,4 and V2N2p3), Joel Pitcairn (514) suggests a way to accomplish the same result (albeit requiring some extra manual keying) making use of a flag. A flag is put in a convenient place inside a loop, and written to cause a branch to interrupt processing code when the flag is set. Then during loop execution, the user keys R/S Stflg n R/S to get a "safe" look at intermediate results. The interrupt processing code should, of course, have in it an INV Stflg n sequence.

Correction: Reinhold's routine (V2N12p5) should have an = before the last x:t, which takes care of the pending arithmetic, and explains the purpose of the x:t. The number of steps is 30, as stated. Reinhold reports that he and a friend have been trying to devise a practical 59 program to calculate all real and complex roots of polynomials up to about degree 15, and are seeking help from the membership.

CROM HIRs (58/59): Use of the HIR instructions (V2N9) is showing up in some of the applications CROM code. T.S. Cox (9) reports an HIR 32 at step 240 of the Leisure Library Football program, and Roy Chardon (515) the use of HIR 8 in Pgm 13 of the Real Estate Library.

More on Unintended CROM Code Execution: Rusty Wright (581) found a code 90 (List) at step 036 of ML-11, and reports that the sequence: Pgm 11 SBR 063 causes total wipe-out (except for the CROM!) when the printer is connected.

Strange LRN Behavior (58/59): Jared Weinberger (221) found that certain manual sequences in run mode followed by LRN would create program code. Specifically: 1) STO, SUM, Exc, Prd, or RCL followed by Ind LRN BST show a code 11 (A); 2) x≥t, x=t, SBR, or GTO followed by Ind LRN BST produce code 22 (INV); and 3) CP Ind Lbl or CP Lbl Ind followed by LRN BST produce code 04 (4); code 02 (2) for the 58.

Printer Character Shifting Using the Fix Function (58/59): T.S. Cox (9) found that he could left-justify 1-3 characters in the Op 4 Op 6 sequence by first putting the display in a fix 6 format. His findings lead to the following generalization for all 4 print buffers: fix n causes an OP 1-4 on the display to be shifted left n places in the print buffer, provided the display is an integer LE 10-n. For example, with turn-on display, key 12 Op 2 Op 5, and see a 9 printed in the 5th position of the first sector. Now, with the 12 in the display, key fix 2 Op 1 Op 5, and see the 9 in the 4th position. Key fix 4...6...8, and see the 9 move successively to the left. Fix n for n odd will, of course, change the code... in this example from the 12 to 0120, which prints 0- .

Writing Good Diagnostic Programs: Reinhold Patzer (689) experienced a special-case fault with his 59 that suggests another consideration in writing rigorous diagnostic routines (V1N4p4): Provide for both manual keying and program execution of functions. The case Reinhold reports may be rare, but it is subtle, and was probably difficult to isolate: On his 59, *"...Reg 65 appears to behave well if used only by the program. If any key is pressed the second bit of the first mantissa digit is cleared"*!. Reinhold didn't say how long it took him to isolate this fault... but suppose it had concerned the mantissa's LSD instead of the MSD... it might well have gone unnoticed indefinitely.

Membership Address Changes: 545: O"Brien Box 127 Somerset, NJ 08873; 658: 20 Confucius Plaza #7C New York, NY 10002; 680: RD 1 Box 286 Newark Valley, NY 13811.

More on Out-of-Bounds Pointer Behavior (58/59): Roger Gentry (398) has pursued the out-of-bounds pointer phenomenon a bit further (V2N12p2) and his results lead to the following observations: If the pointer is positioned within the first octet of steps beyond the current partition, repeated keying of LRN will eventually put the machine into learn mode. The initial position of the pointer within the octet produces different results: For end-of-partition (EOP) + 1. 2 LRNs show step EOP; for EOP + 2, 2 LRNs show step EOP + 1; for EOP + 3, 2 LRNs show EOP + 2; EOP + 4, 5, 6 repeat this pattern, except that 3 LRNs are required, and EOP + 7 and 8 take 4 LRNs. In each case, if a displayed step is out of bounds, its value is always 00, regardless of what may have been written there. An SST switches to RUN mode, but BST appears to operate normally from EOP + 2 to EOP + 1 to EOP. With EOP + 1 or 2 in the display, LRN SST

effectively executes the 00 (or appears to). "Real" code at steps EOP + 1 or 2 is not affected by any of these maneuverings. All of this may have something to do with the code execution buffer (V2N10p5). If so, it would seem that repartitioning so as to just barely place out of bounds the register whose contents is currently in the code execution buffer does not completely cut of access to this buffer.

Fractured Digits Revisited (58/59)

Fred Fish (606) came in second to Jared in the fractured digits race (by letter), but has a technique that doesn't require printer connection, and makes possible the generation of at least 3 fractured digits in the display. Like Jared's, Fred's approach builds on Rusty's, but the key element is use of the SST function. Following a display position notation of 0-11 analogous to the 0-13 one adopted for the SR-52 (V1N2p5), it is possible to fracture positions 1, 2, and 7 with one sequence, and position 5 with another. Partitioning determines the position 7 character in the same way as explained on V3N1p1; similar partitioning dependency, but different rules determine position 5; positions 1 and 2 can be made ", blank, or unfractured numeral by specific combinations of flags 0, 1, 5, and 6. Fred's sequence for fracturing position 7 is: Stflg SST, with a full display, starting in learn mode at the last step of the current partition. The same sequence for fracturing position 1 with a " includes the setting of flag 6; setting both flags 1 and 6 changes the " to a blank. With only flag 5 set, the " goes in position 2, and setting both flags 0 and 5 changes the " to a blank. Following the SST with = produces a soft 6. Setting only flags 5 and 6 produces the " at positions 1 and 2. Replacing Stflg SST with fix SST produces the same fracturing, but after the =, a soft 8 is produced; Dsz SST produces ms 0.0 ds 90 where ms is the mantissa sign and ds is the decapower sign of the initial display; and Ifflg SST produces ms 0.0 ds 80. GTO, Lbl, or SBR SST fracture a position 5 digit, depending on the following partitioning: 239.XX produces a blank, 319.XX a ', 399.XX a °, 479.XX a -, and (for the 59) 959.00 a " . The execution of the SST in Fred's approach appears to move the pointer "somewhat" out of bounds: it takes 2 successive LRNs to get into learn mode, but the step displayed is the "proper" last one (see V3N1p5).

I'll wind this up with a collective hats off to Rusty, Jared, and Fred for getting us started at 58/59 fractured digits, and look forward to more inputs from others on this lively topic.

Volume 3 Number 2 48/39 February 1978

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

Practical Application of Obscure but Powerful Programming Features

In asking the question: has anyone found a practical use for having a subroutine call itself? Cliff DeJong (290) suggests a broad topic for membership exploration: Given one of the more powerful but seldom-used machine capabilities, show how it can enhance the solution of a practical programming problem. Several somewhat obscure 58/59 capabilities come to mind: 1) Use of all 6 subroutine levels; 2) Use of both an indirect conditional and its indirect transfer together, i.e. Dsz*ab*cd; 3) Use of the list, trace, and Op 8 functions under program control; and 4) Use of HIR operations to alter stacked operands before pending arithmetic is executed. Members are invited to address these and to suggest other related topics covering any of the TI PPCs.

I'll lead off with some thoughts concerning Cliff's question. Given the architecture of the TI PPCs, I can't think of a practical application where having a subroutine call itself is an advantage. Sequences of the form: ...Lbl a ...b ...Lbl b ...rtn are indeed useful (see the V2N5p5 program), but amount to 2 routines sharing some common code, not one routine calling itself. A routine of the form: ...Lbl a ... a ...rtn calls itself, but without further specification, would not terminate. It could be made to terminate by writing something of the form: ***Lbl a*** *seq1 ifflgx b stflgx a* ***Lbl b*** *seq2 rtn.* Calling it with Flag x unset would cause the code designated by seq1 and seq2 to be executed as: seq1 seq1 seq2 seq2; calling with Flag x set would produce: seq1 seq2. More flags could be introduced to produce more variations, but I suspect the overhead of flag testing and setting would make straight forward calls to separate subroutines more attractive. Anyone else care to address Cliff's question?

Efficient Print Code Storage

Clive Durbin (618) has found a HIR application which significantly minimizes data storage and processing requirements in many cases. His approach takes advantage of the differences in how Op 4 and HIR 08 treat display values during transfer to the print buffer. While Op 4 copies the 8 LSDs of the integer part of the display into the 8 LDSs of the print buffer (HIR 8), the HIR 08 function copies an entire real, as formatted in the display register. Thus only the print code contained in the integer part of the displayed mantissa is used to generate a character string with an Op 4, while as many as 8 LSDs (regardless of decimal point position) are used with HIR 08 storage. Using Op 4 to prepare one character string for printing, and HIR 08 on the same number for another

not only saves the unpacking required in the V2N11p6 approach, but allows for overlapping that can enable one real to prepare 2 strings of up to 4 characters each. For example, Op 4 Op 6 on 916323117.4437 produces the tag DONE, while HIR 08 Op 6 on the same number produces NEXT. Note that the leading 9 serves as a dummy filler to force right justification within the 13 mantissa places, and that because of the odd number of mantissa places, overlapping with Op 5 printing would be difficult at best. However, there may be useful strings of characters some of whose print codes can be split and repaired to form other useful strings. For example, 1624222437.253 produces DIGIT with Op 1 Op 5, but VGW÷π with HIR 05 Op 5 (sorry, I couldn't think of a more useful example!). Members are invited to share their best (most practical) Op 4/Hir 08 Op 6 and Op 1-4/ HIR 05-08 Op 5 creations.

Recording Current Machine States (52,59)

In many practical applications it would be helpful to be able to record on mag cards such things as flag status, display format, angle mode, partitioning, printer connection, CROM module number, the contents of the HIRs and the T register ... prevailing conditions which the user might need to manually reinitialize every time he reads a particular card. Converting such information into recordable data is, for most of the above, a fairly straight forward exercise, but doing it efficiently may be somewhat challenging. For example, 10 registers could be designated to hold flag status: each stored with a 1 for set, 0 for not set, but this might be unacceptably wasteful of data registers. So it is attractive to consider data packing as an alternative. The following sequence stores the status of all 10 flags in Reg 1, taking advantage of the machine interpretation of flag 10 as flag 0 (TI-59): *... 0 STO01 10 STO00 1 STO2* ***Lbl1'*** *CLR INV Ifflg*0 2' 1 Lbl2' X RCL2 = SUM01 10 Prd2 Dsz0 1' ...* and the following takes the value in Reg 1 and sets all 10 flags accordingly: *... CP 10 STO0* ***Lbl3'*** *RCL1 ÷ 10 - Int STO1 = x=t 4' Stflg*0* ***Lbl4'*** *Dsz0 3' ... .* See V1N5p6 for a display-format determining routine, V2N11p6 for angle mode indicator, and V2N9p2 and V2N10p3,4 for printer connection sensing. Partitioning can be decoded with: *... Op 16 INV Int X 10 + 1 = Int STO1 ...* and set with: *... RCL1 Op 17 ...* . Picking a suitable approach to saving the contents of the HIRs is somewhat frustrating: There ought to be a straight forward way to loop through a basic sequence 8 times, but there doesn't seem to be since the HIRs are not indirectly addressable under program control. The following sequence will transfer the contents of the HIRs to Reg 1-8 by an iterative process, requiring dynamic code modifica- tion (V1N2p3), but not efficiently: *... 10 Op 17 10.82 S99 8 STO0* ***L1'*** *10 Op17 1 SUM99 6 Op17 GTO 166 ... 168: STO*0 Dsz0 1' ...* . The straight forward: *... HIR11 STO1 HIR12 STO2 HIR13 STO3 HIR14 STO4 HIR15 STO5 HIR16 STO6 HIR17 STO7 HIR18 STO8 ...* is both shorter and faster.

Members are invited to find and share better approaches, and to suggest other things worth recording; efficient recording utilities will be valuable to many users.

Sorting and Searching: Some Popular Applications (59)

While member feedback on the V2N8p1 and V2N11p2,3 articles has so far been slight, there is growing interest in developing efficient approaches to 2 related topics: 1) Addressing block-stored data by arbitrary tags, and 2) Sorting data by magnitude. Mechanization of both topics reveals machine as well as data dependency, and since practical implementation concerns more than just a few data, the discussion which follows centers on the TI-59/PC-100A.

Addressing Block-Stored Data by Arbitrary Tags: Bob Anderson (506) poses a common business problem, whose efficient solution should be of considerable interest to many users: A collection of entities, which I refer to here as records (machine-represented by specified numerical values), is to be stored in a specified block of data registers, each record identified by a unique tag or key (also machine-represented by a specified number) such that the required code and the execution time required to address each record from its key are minimized. The special case where keys match one for one the addresses of allocated data registers is simple enough to mechanize efficiently via indirect addressing, but it precludes the user from choosing "natural" keys: numbers and numerical representations of characters directly associated with the stored records. So let's look at a realworld situation which Bob poses: 20 departments in an organisation are identified by the numbers (keys): 2,4,6,8,9,10,13,17,18,22,23,25, 29,31,32,33,34,51,72, and 73, each of which is associated with a labor rate (record). The records are to be stored in a concise block of data registers (not scattered throughout the Reg 2-Reg 73 range), and be efficiently addressable by their keys. A straight forward approach would be to compare an input key with all stored keys, one at a time until it is matched, at which time processing is diverted to appropriately initialize a pointer. But this would be prohibitively costly in both code and execution time. Another approach might be to try something along the lines of Ordered Hashing (V2N8p1), which Bob experimented with, using the ML-15 random number generator to serve as the hash function. This is an improvement, but is still a bit slow, and requires handling quite a few "collisions" (2 or more keys produce the same hash (register) address)... a problem not likely to be solved by trying to find a better hash function. But assuming that the keys are not to be changed, in some cases advantage can be taken of the pattern they form. In this example, all but 3 of the 20 keys can be matched one for one with a register in the 1-34 address range, which might be an acceptably concise block. Each of the remaining keys: 51,72,73 should then be matched with one of the 14 unassigned registers in the 1-34 address range, and use made of the remaining 11 for program scratch. But what's the best way to match the remaining keys? It is tempting to look for a special no-collision hash function, which might be mechanized along the lines of the following algorithm:

If K ≥ 51 set h(K) ← (K mod M) + 1
Else set h(K) ← K

which will do the job with M=47, and in English says: If an input key (K) is greater than or equal to 51, evaluate a hash function h(K) by adding 1 to the remainder modulo M (the remainder in integer form resulting from the division of K by M) of K.

If K is less than 51, use K itself as the hash function value. In either case, the value h(K) points to the desired register. The number 47 was found by trial and error to satisfy the relations h(51)≠h(72)≠h(73), all in the 1-34 range of unassigned registers. The routine: ***LblA*** *STO1 x:t 50 x≥t 1' RCL1 ÷ 47 = INV Int X 47 = fix 0 EE INv EE STO1 Op21* ***Lbl1'*** *rtn* accepts one of the 20 keys in the display, and following a call to A, produced the corresponding h(K) in Reg 1. Incidently, the *fix 0 EE INV EE* is required for rounding, and although fix 0 D.MS works just as well, it takes about 4 times as long to execute. The fix 0 can be eliminated if this routine is always called with a fix 0 display.

But it turns out that for only 3 keys, a straight forward matchand-process approach is considerably faster, and not much longer: ***LblA*** *STO1 x:t 50 x≥t 1' 51 x=t 2' 72 x=t 3' 26 STO1 rtn* ***Lbl2'*** *5 STO1 rtn* ***Lbl3'*** *27 STO1* ***Lbl1'*** *rtn.* This exercise illustrates an important programming precept: Don't let exotic schemes close your mind to simpler ones that are better (unless your objective is to make an exotic scheme work!). One more thing to keep in mind when approaching the problem of how to best process arbitrary keys: It may pay off to spend a fair amount of effort looking for patterns which can be matched by simple non- or low-collision hash functions. There is a straight forward mathematical procedure for fitting n-1 data points with an nth degree polynomial, and while such an approach is impractical for large n, it suggests a compromise: Try out various combinations of simple mathematical functions: trig, log, arithmetic, etc; with perseverance and a little luck you may be able to solve the collision problem.

Sorting Data By Magnitude: In the previous problem, the user doesn't care where each record is stored, so long as it is readily accessible by an arbitrary key. Sorting data by magnitude is a different kind of problem, and arises any time the user wishes to start with an unordered sequential set of records and put them into ascending (or descending) order. While these records could also have arbitrary keys, such are not necessary to illustrate the sorting techniques discussed here, and in the interests of simplicity a record's key will be considered identical with the address of the register in which it resides. There are a few data dependencies to keep in mind when choosing an approach: 1) The initial ordering, 2) The number of repeated values, and 3) The number of data. These combined with machine idiosyncrasies make the choice of approach consequential. With no packing, about 100 data is a practical limit for the 59 to process, and for quantities in this neighborhood a variant of the so-called Shell Sort method does reasonably well. The program that follows was mechanized from an algorithm appearing in **Popular Computing** (Jan 78 p5). Members wishing to examine Donald Shell's method in detail will find a technical discussion in Vol III pp 84-95 of Knuth's **"The Art of Computer Programming"**. Use is made of the HIR operations (V2N9) to maximize data capacity, and as you may have noticed earlier in this issue, the mnemonic HIR has been shortened to H. A reverse-order input of 99 records takes about 21½ minutes to sort; a sample of 99 random numbers produced by ML-15 took 26½ minutes

TI-59/PC-100A Program: Shell Sorting of up to 99 Data Ed

User Instructions: Key number of data (n), press E; key records (ri), press R/S, for i=1,2,... n. Execution begins following the input of rn, and when the program halts, Reg 1 - Reg n contain the input records sorted low to high, the contents of which may be displayed by successive R/Ss if a printer is not connected, or are automatically printed if it is.

Program Listing:

```
000:  Lbl E x:t 10 Op17 CMs x:t HIR 8 HIR 7 STO0 R/S STO*0 Dsz0 015 HIR 17 ÷
      2 = Int
028:  HIR 7 CP x=t 104 1 HIR 6 HIR 18 – HIR 17 = HIR 5 HIR16 HIR 4 HIR 14 STO
      0 RCL*0 x:t HIR17 +
059:  HIR 14 = STO0 RCL*0 x≥t 090 x:t STO*0 HIR 14 STO0 x:t STO*0 HIR 17
      HIR 54 1 x:t HIR 14
087:  x≥t 049 1 HIR 36 HIR 16 x:t HIR 15 x≥t 045 GTO 022 1 INV List STO0
      RCL*0
111:  R/S Op20 GTO 109
```

Sorting permutations of a fixed quantity of specified values by magnitude is a special case of this second type, which Lou Cargile (625) has been exploring using several of the TI/HP PPCs to effectively perform card shuffling, dealing, and arranging for the game of Contract Bridge. His 59/100A program which follows combines a number of illustrative approaches and techniques, and will be a tough challenge to anyone aspiring to write a better one.

TI-59/PC-100A Program: Bridge Deal Lou Cargile (625)

User Instructions: Key RN seed < 1, press E; (optional): key another seed < 1, press D. To print one deal, press A; to print n deals, key n, press B.

Program Listing:

```
000:  Lbl E' RCL*20 Op0 Op1 Op5 Op0 Op5 1 INV SUM20 rtn Lbl A RCL19 SBR245 39
028:  STO17 1 SUM25 RCL25 Prt CLR INV Stflg2 RCL19 X RCL26 + RCL27 = INV Int
051:  STO19 + RCL16 = INV Int X 52 = Int STO11 ÷ 13 = Int STO12 + 1 = STO0
      RCL11
080:  + 1 - RCL12 X 13 = STO13 RCL*0 ÷ RCL13 INV log EE STO13 INV EE = INV
      Int
106:  X RCL13 = EE x:t CLR RCL13 ÷ 10 = x:t INV x≥t 036 x:t INV SUM*0
128:  x:t 1 INV SUM17 4 SUM0 x:t SUM*0 RCL17 ÷ 13 = INV Int x:t 0 INV
150:  x=t 036 x:t Stflg2 8 STO0 E' 0 STO24 4 STO9 RCL*0 x=t 211 log EE Int
      STO15
176:  INV EE INV log EE INV SUM*0 CLR RCL9 + 61 = STO21 RCL15 + 47 = STO23
201:  RCL*21 + RCL*23 X 100 = Op*9 SUM24 1 INV SUM0 Dsz9 166 RCL24 x=t 236
228:  Op05 4 SUM0 GTO 160 Adv Ifflg2 333 Dsz"22" A R/S STO19 1 EE 13 INV
252:  EE ÷ 9 = STO1 STO2 STO3 STO4 3 STO47 4 STO48 5 STO49 6 STO50 7 STO51 10
      STO52 11
285:  STO53 12 STO54 2.01 STO55 25 STO56 34 STO57 26 STO58 13 STO59 9821
      STO26 .211327
326:  STO27 69 STO20 rtn RCL17 INV x=t 036 INV Stflg2 E' GTO 160 Lbl B STO22
      GTO A
352:  Lbl D STO16 16 Op4 RCL16 GTO 395 Lbl E x:t 6 OP17 CMs 7 Op17 Op0 RCL60
      Op2
381:  RCL61 Op3 Op05 17 Op4 Adv x:t STO19 Op6 Adv R/S
```

Pre-stored Data:

```
60:  143524 1622170000 36000000 23000000 16000000 15000000 4317363700
67:  3632413723 1713363700 3132353723
```

Tips and Miscellany

A Safe, Handy Hardware Interrupt (58/59): In cases where no CROM program is called within a loop, George Hartwig (638) has found a method paralleling the SR-52

D-R switch application (V3N1p4): a dummy call to a CROM program places at a safe position in the loop works nicely with a manual RST to cause a transfer to step 000 of user memory. For example, write: *000: R/S **Lbl A** 20 STO0 **Lbl1'** Dsz0 1' Pgm 7 SBR005 GTO A.* Press A, and after an arbitrary time interval, press and hold down RST until execution halts. No matter when the RST is pressed, the halt occurs at the step 000 R/S, following a completed Dsz cycle. It appears that is doesn't matter at what step the CROM program is called; so long as the RST is depressed during the call, no CROM code is executes. If the CROM call is to an undefined label, an error condition is set, but transfer to step 000 of user memory still occurs. But for practical application, it is best to call a CROM program at a rtn (as in the above example), since this minimizes execution time (about 200 ms) and doesn't let the CROM code do anything during times when the RST is not depressed.

Use of p41 to Make Lbl Lbl Tricks Work for the 58/59: Rusty Wright (581) discovered that the SR-52 LBL LBL tricks can be made to work on the new machines if each Lbl Lbl sequence is preceded by the SST pseudo (p41). For example, Jared's flag reversal routine (V2N10p2) for the 52 can be written: *...Ifflg0 p41 Lbl Lbl INV Stflg0...* for the new machines. Carl Paulson (854) has been exploring other sequences with p41 executed under program control, and finds that the first R/S or any number of P31s following one or more p41s is ignored. All this added to the unorthodox manual use of SST in creating fractured digits (V3N1p6) should help to encourage further exploration for new SST/p41 uses.

PC-100 to PC-100A: Mike Brown (128) reports that he turned his PC-100 into a PC-100A by positioning the 52-56 switch halfway between the 2 intended settings, effectively creating the "other" position on the PC-100A. The only problem Mike encountered was that sometimes the machine *"... has trouble printing a line of 20 of the same characters"*, which he suggests may be due to a less robust power supply for the PC-100.

56/57/58 Program Exchange: Dave Johnston (5) has a new catalog dated 1 Feb 1978 listing 26 math, 8 statistics, 9 operations research and simulation, 24 physics, 10 other physical sciences, 24 engineering and technology, 3 live and behavioral sciences, 13 games, 6 finance, and 1 general info-test routine... programs written for the SR-56, of which 34 have been translated for the 57, and 15 for the 58. A few more programs in scattered categories were written for only the 57 or 58. Send Dave a SASE and 15¢ for the catalog. He continues to provide program copies at the modest rate of 5¢ per page.

CROM HIRs (V3N1p5): Several members have noted that T S Cox probably meant to refer to the Blackjack program (11) of the Leisure Library, and that while step 240 is a code 82, it is part of a GTO address, and was not intended to be a HIR operation. On the other hand, Roy Chardon's reported use of H8 in Pgm 13 of the Real Estate Library was intended (HIR 8 at step 183 and HIR 18 at step 219). In order to avoid confusion, members are asked when discussing CROM code to indicate whether a stated sequence is intended or unintended. Intended code is defined as that what the printer would list, addressed continuously from start to finish; unintended as printed starting from an intermediate step which separates the elements of a composite instruction.

Volume 3 Number 3 48/39/38 March 1978

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

Book Review: User Survival Guide for TI-58/59 Master Library, by Fred Fish (606), $10.25 postpaid to members, 119 pp, copyright 1978

Fred has put together a comprehensive package designed for the serious user of the ML programs, who wants to know how they work, how to interface them with his own programs, and how to take shortcuts. He brings to the user's attention conditional caveats, and pitfalls to avoid, and details specific program calling sequences to optimize register usage, execution speed, and required user code. This work appears to be technically sound, includes listings, flowcharts, and even a few cartoons, is well-proofed, and I am pleased to give it my endorsement. Include your Club membership number with your order, and ask Fred about quantity discounts.

A Review of the TI-57 Vis-a-Vis its Contemporary PPCs

The 57 has a few unique features which fall into both + and - categories when compared with other PPCs. This review is intended as an aid to 57 users as they either start from scratch, or attempt to translate other PPC programs, and to acquaint non-57 users with this somewhat dif ferent machine.

It is interesting that the low-end TI PPC's unique features are common to some of the high-end HP's architecture (67 or 97):

1) the numerals 0-9 are used as labels,
2) all composite instructions are merged,
3) the step following a conditional can be any instruction,
4) during execution, the display flickers intelligibly (which the 67/97 can be made to do under certain circumstances).

Perhaps the most critical limitation is the comparatively large amount of register sharing. While the number of data registers (8) puts the 57 into the same league as the 56 or HP-25, shared use of some of them is excessive: 1) The last 2 (of a max of 4) pending operations go into Reg 5 and 6, 2) Reg 7 is also the T register, and 3) The Σ+ uses all the data registers except Reg 6. However, the latter results from a positive feature: the Σ+ function processes 2 variables at a time. Another important limitation is the 8-place display (with 3 guard digits). While this may be acceptable for most applications, the user should bear in mind that actual accuracy may in some cases be lower than expected (1 INV lnx produces e to only 9 places). Execution speed of the 57 is generally slower than that of the other TI PPCs, especially for the trig and exponential functions. A realistic comparison of the 50 merged steps with the 56's 100 unmerged ones will be deferred until a few representative challenging test cases surface.

However, as a starter, when I attempted translation of Dix Fulton's V2N10p1 program, it overflowed by about 20 steps.

A few programming tips come to mind: 1) Instead of inputting via a STO 1 R/S STO2 R/S ... programmed sequence, write STO1 STO2 ... and execute with the SST key; 2) Take advantage of the anything-goes step following a conditional, where such functions as R/S, rtn, Pause, +/-, etc may be better than GTO n; and 3) If you run out of user-defined labels, SBR 2nd nn from the keyboard (but not under program control) will start execution at step nn. 57 users are invited to comment and share their tips, inventions, and discoveries.

Fractured Digits: More Control (58/59)

Roger Gentry (398) has discovered some new ways to produce fractured digits, one of which leads to a capability to display a few selected characters with or without specified numerals. For example, with a 59 at turn-on or a 58 at turn-on and a 459.0 partition, key: *GTO 441 LRN List 0 0 0 0 0 LRN Pgm 11 SBR 444 R/S LRN*, and see 2 ° symbols looking like a pair of eyes. The machine at this point appears to be in LRN mode, expecting a 2-digit register address. Pressing any non-numeral except LRN writes code in the usual way; pressing a numeral "closes" the eyes, producing 2 - symbols in addition to writing a number code. Press LRN BST BST and see the keycode for the numeral you just keyed, or key another numeral than BST, and see the merged code for both numerals. Other (but not all) permutations of fractured digits in 3 consecutive display positions can be produced similarly, following rules based on this discovery of Roger's: The current step number in the user memory IAR which prevails when the sequence: *Pgm 12 SBR 999 R/S LRN* is keyed manually, determines what characters are formed at display positions 8, 9, and 10 (using the 0-11 full display convention). Roger observed that when a step is converted into 3 such codes by dividing it by 8, interpretation of the quotient, and remainder in integer form a V1N2p5 character code produces the expected characters. In the above example, the call to Pgm 11 is made at user memory step 447. Division by 8 produces 55 remainder 7 which are the codes for ° °blank. But there are o ther requirements to be met to assure the desired display. If we label the octet containing the "launch" step (in this example step 447): AB, CD, ... OP (as in V1N1p5) the following rules appear to be necessary to follow to produce just the 2 eyes: A must be zero, B can be anything, C must be non-zero, and OP can be anything. Note that MN and OP are set to 72 and 42 respectively, following execution. Non-zero EF, GH, and IJ can produce up to 6 numerals preceding the eyes; non-zero A and C show up to the right of the open and closed eyes, respectively; fix n, n: 1-8 positions a decimal point in accordance with n; launch at other than the last step in the octet produces different results ... there seems to be no end to the variations. Members are invited to explore further, and to report useful (or what may appear to be potentially useful) creations.

Label Search Methods: A Comparison of TI/HP Machines

John Hirsch (763) notes that while considerable attention has been given to the pros and cons of AOS vs RPN, label addressing comparisons have received little attention. John points out that the TI machines have absolute and label addressing, contrasting with the newer HP machines' relative and label addressing.

Perhaps a definition of terms at this point would help to avoid confusion. In past issues of 52-NOTES I have used the term relative to describe addressing within relocatable code, and absolute for unrelocatable code. John uses the term relative with the HP machines to mean an addressing method whereby an address is calculated by adding/ subtracting an integer value to/from the address of the currently executed instruction. In other words, the program counter (or what I prefer to call the IAR (V2N4p5)) is either incremented or decremented by a specified amount. It is in this same way that the word relative is used to describe general purpose computer addressing, and since it is quite possible that a new TI PPC (or personal microcomputer) might have such addressing, henceforth I will use the term relative addressing with this meaning, and as John suggests: label addressing for what I used to call relative addressing.

HP-67/97 relative addressing is backwards only (a negative value stored in the I register is added to the contents of the IAR), so for forward transfers, the user must count the number of steps back through 000, 224, 223, ... to the desired step, and then provide for storage of the negative of this count in the I register. The primary advantage of this scheme is that it shares the same speed enhancement and memory-region independence of the TI machines' absolute addressing without the unrelocatability restriction. Another important difference is the label search mechanism: The HP machines start a search downward from the instruction starting the search. This makes if difficult to optimally locate subroutines called throughout a main program, but makes it possible to use the same label more than once, and expedites transfers within code located low in memory.

John made some label-search-execution time comparisons among the SR-52, SR-56, TI-57, TI-59, and HP-29c machines, and reached the following conclusions:

1) There is really not much difference in label search time when differences in total program steps are considered, although the TI-57 came out best.
2) The HP disadvantages offset the advantages.

and suggests for 3rd generation machines a key which gives the user the option of specifying label search to be forward, backward, or from top to bottom.

Other members are invited to comment, and contribute further to this topic.

Routines:

Fibonacci Number Generator Revisited (V3N1p3): Upon applying his routine to large numbers, Joel found that correct answers could be extended a bit by calculating the $(1 + \sqrt{5})\div 2$ constant (known to mathematicians as Tau or Phi, or popularly as the Golden ratio as $(1 + (1 \div \sqrt{.2}))\div 2$. I will refer to $(1 + \sqrt{5})\div 2$ as t_1, and $(1 + (1\div\sqrt{.2})) \div 2$ as t_2, and Tau in general as t. Incidently, some interesting things about t are due to the relationship: $t^2=t+1$, which may also be written: $t=(t-1)^{-1}$. It turns out that t_2 is a better approximation than t_1 by one place, because of machine rounding and truncation. As calculated, t_1=1.618033988749, t_2=1.618033988750, but since t=1.61803398874989... t_1 is good to only 12 places, while t_2 is good to 13.

Efficient I/O Handling For Engineering Problems (58/59): Clyde Durbin (618) found that the nature of some of his engineering problems suggested some useful tricks/shortcuts, which should also be helpful in related applications. One situation which arises often concerns the user's desire to specify printing and tagging of a few outputs out of many possible ones, following initial processing. The sequence: ***Lbl A*** *STO0 RCL*0 HIR8 GTO*0* when called by n A, where n is a register address, will cause code to be executed and tagged results printed, both specified by the contents of Reg n, provided Reg n is initialized with: *sss.00tttttttt* where *sss* is the step where specified processing is to begin, and *tttttttt* the print code for the desired tag. The step sss sequence ends with *... Op6 R/S*. Clyde has also found that in cases where 2 or more input parameters are in mutually exclusive numerical ranges, they can all be entered with a single user defined key. The program first compares an input with prestored boundary values, and branches accordingly.

Copying the Display into the T Register (56,57,58,59): There are times when it is useful to effectively store the display in the T register without changing the display. Jared Weinberger (221) suggests: *... (CP + x:t)...* . Does anyone have a shorter way? *(...CP + x:t =...* doesn't count!, and for the 57, *STO 7* does the trick).

Digit Reversing (V2N11p6): Reinhold Patzer (689) has trimmed Jared's by 3 steps and one register with: ***Lbl A*** *(CE ÷ 10 Prd 1 - INV Int SUM1) INV x=t A (Exc1 X 10) rtn* and notes that both routines will handle up to 13-digit numbers.

Tips and Miscellany

A New Hidden Facet of the Old Machines (52,56): Ken Whipple (849) discovered that his SR-52 dimly displays a numeral representing the number of digits to the right of a displayed decimal point in the otherwise unused decapower LSD position for a real in fix format. Ten digits to the right (the maximum) even shows up as an A (as if the machine knew it should count in the hexadecimal number base)! I confirm Ken's findings, and add that the SR-56 counts the same as follows: ' °- blank 5 6 " blank ' A. So here's one way to get fractured digits on the 56. The new machines do not seem to respond in this way, but perhaps there are related frontiers waiting to be explored via darkened room and magnifying glass.

Extra Memory for the TI-58: Bob Edelen (100) reports that he has successfully added 2 TMC 0598 chips (order from TI per V2N5p3 for the TMC 0599, omitting Riggs' name) to his 58, effectively converting it into a 59 without mag card read/write. Bob has a supply of these chips, and is prepared to install them for 58 owners. Write Bob at his business address: 5440 Roslyn #285 Denver, CO 80216.

Data Entry Sensing (V2N7p5): Reinhold Patzer (689) brought to my attention the best approach I've seen yet for program-sensing of data entered from the keyboard. The method is due to Heinrich Schnepf (376) which he describes (in German) in V2N2p35 of his **Display** newsletter, and is simply to follow a programmed data-entry R/S with a decimal point and a zero test. It appears to work for all reals so long as the display is zero or hard at the R/S, and is left soft following a datum entry; a keyed zero is treated as no entry. This trick works because the decimal point zeros a hard display, but does not alter a soft one.

A New R-P Convention (57,58,59): J.R. Merrill (693) notes that the new machines return positive angles in the 0-270° range, and negative to -90°, unlike the older TI and all the HP machines, which return ± 180°, for rectangular to polar conversions. Which is better? J.R. prefers the latter, but there may be applications favoring the former. Page V-31 of the 58/59 owner's manual notes the 4th quadrant negative angle convention, but doesn't say why it was chosen. The 57 manual doesn't appear to address the matter of returned angle signs at all.

Fast Constants: When deciding whether to create or store a constant, if execution speed is critical, note that creating a digit string with more than 3 elements using program code takes longer to execute than a RCL.

Runaway Mag Card Drive Motor (59): John Hirsch (736) reports occasions when following card drive, the motor can only be stopped by turning the machine off. The owner's manual (p VII-8) states that keying R/S should stop a running motor, but implies that it may be "normal" to have to turn the machine off in some cases. Anyone else experiencing John's problem, let me know.

More on Pgm mm R/S (V3N1p3): John has found that there are indeed applications for sequences of the form: Pgm mm R/S in user programs. After a CROM routine in Pgm mm has been called by a user program, the effect of Pgm mm R/S is to resume CROM execution at the step following the last rtn encountered, just as a plain R/S does during keyboard interaction with a CROM program. A CROM pointer maintains this restart address through intermediate user-code execution, but not following a manual or programmed call to another specified Pgm label or step number. For example, write: ***Lbl A*** *3 Pgm 2 A 1 Pgm 2 B R/S* ***Lbl B*** *Pgm 2 R/S R/S GTO B*. Press A, see 1; key a datum, press R/S, repeat for say 3 data. Now go elsewhere in user memory, write a small routine (that doesn't clobber the ML-02 registers, but may use all 6 subroutine levels), run it, then key the next datum for input to ML-2, and press B; key a few more data with R/Ss and find that all data were stored sequentially as expected in Reg 8, 9, ...

Creating Large Mantissas (57,58,59): For all 3 machines, creating a number in scientific notation whose mantissa is larger than 8 places can be done, but requires a few steps. For the 57, one way is to store the first 8 digits in a convenient register, SUM up to 3 more (appropriately scaled), then Prd 1 EE dd, where dd is the desired decapower. For the 58/59, the SUM step can be skipped for mantissas less than 11, or included for mantissas of 11 to 13 places.

Overflow and Underflow (57,58,59): The V2N1p3,4 discussion applies to the new machines, for the most part, but with a few exceptions. For all 3, display rounding of large numbers to 8 mantissa places corresponds to 10 places for the 52 and 56. When the 57 transfers any conditionally overflowed number from a data register to the display, it is converted to the upper-limit overflow number: 9.999999999D99. So operations performed on a conditionally overflowed number should be via register arithmetic.

Membership Address Changes: 128: Back to original address; 452: 3812 Kendate Dr Gautier, MS 39553; 553: 11 Plumeria St APRA Hts FPO SF 96630; 664: 100 Barton St #10 London, Ont N6A 1N2 Canada.

Absolute Code Execution Speed (52,56,58,59): Roger Gentry (398) notes that absolute transfers near the top of memory are faster than those near the bottom (high addresses). The effect is the same as for label transfers, but is not so pronounced. Short Dsz loops placed within a 1-register octet of steps run a bit faster than when straddling 2 or more registers.

HIR Operator Modifications (58/59): Rodger discovered that under certain circumstances HIR operations in the middle of pending arithmetic can change the originally designated arithmetic operators. For example, *n - HIR11 HIR1 = produces 2n*, and *n ÷ HIR11 HIR1 = produces* n^2. This behavior suggests that numbers pushed into the HIRs via pending arithmetic may not be so "clean" as once thought (V2N9p1).

More on CROM HIRs (V3N2p6): The Leisure Library program which T.S. Cox meant to refer to is Craps (13), which contains a "real" HIR at step 240. I say real, not "intended", because my definition for the latter is poor in this example: The only way this HIR operation could do anything is via a SBR 240 call, which would add the display value to the contents of HIR 2; SBR CLR R/S would add zero to HIR 2, and I don't see any other way for the program to even get to step 240. Also, there doesn't appear to be any use made of the contents of HIR 2. I make this assessment based on a listing kindly provided by T.S. (I'm still waiting for my free copy of the Leisure Library).

Mag Card Tips: Jerry Johnston (493) suggests folding a strip of paper around a mag card before inserting it into one of the TI wallets, to facilitate extraction, and recommends Pelican #17 black ink for permanent card marking.

More on the Hardware Interrupt (V3N2p5,6): Roy Chardon (515) adds an improvement to make the interrupt handler module-independent: Make the CROM call to Pgm 1 SBR 021, since every CROM appears to have a rtn at that step. Roy also warns that the manual RST does, of course, reset all flags and clear the subroutine-return register. The R/S at step 000 of the V3N2p6 sequence serves no purpose, and should be deleted or replaced by GTO N or GTO nnn, where label N or step nnn head an interrupt processing sequence, if such is not to start at step 000.

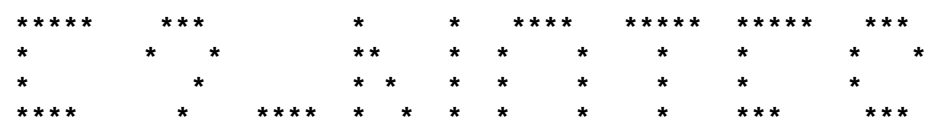
CROM Pause: The statement concerning step 353 of Pgm 19 in V2N8p6 is incorrect. That step is a rtn, not a pause. Code 66 in CROM, either intended or unintended executes as pause.

DC Adapter: Mack Maloney (246) notes that TI's 12 volt DC to 120 volts AC convertor (model DC 9105) was designed for use with the 52,58, and 59 (but would presumably work with other PPCs), and is available at the Washington DC TI Exchange Center.

Quick Manual Flag Tester (58/59): Nolan Tobias (853) notes that the V1N4p6 method for the 52 can be shortened to Ifflgn N, where N is an undefined label.

The R/S' Function (58): J.R. Merrill (693) notes that the 58's equivalent of the 59's Write function appears to execute as Int, and Jared Weinberger finds that when used as a label, R/S' behaves like R/S (V2N11p4). Roger Gentry (398) has found that a prevailing fix n condition, n: 1-8, followed by R/S' multiplies the integer part of the display by 10^n.

CLR/CLR' Difference (58/59): Joel Pitcairn (514) wonders if anyone has found an application where CLR and CLR' execute differently.



Volume 3 Number 4 48/39/38 April 1978

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

Detecting Machine Hardware Changes

As was the case with the 52 and 56, TI is making hardware changes in the new machines without changing names or adding revision identifiers. Fortunately, the date of manufacture is stamped on the back of each machine. (As I recall, one of you phoned this information to me, but I neglected to write your name down). By relating machine characteristics to date made, perhaps we can determine when detectable mods were made. The date is coded in 4 digits following the letters LTA (DTA for older machines made in Dallas rather than in Lubbock) in an aabb format, where aa is the week and bb the year. If one of you electronics-hardware experts will volunteer to be the focal point, I'll invite members to describe their machines (operational behavior, component descriptions, PC board design, etc) to you, and publish your findings. Dave Leising (890) notes a component difference in late-model 59s which he identifies with a card read/write improvement. I confirm Dave's identification of an added transistor, and note also the addition of more resistor/capacitor-like components, and a different PC board on a 59 made the 49th week of 1977, as compared with one made the 24th week. The newer 59 does indeed read and write mag cards with fewer errors, and displays a brighter C when calculating (which may not be an improvement if this requires more power).

The ability to relate machine configuration to the date made should help the buyer find the best machine design currently available as well as give the user a better understanding of his particular machine's behavior.

Precision and Accuracy

In the context of PPC calculations, precision is the degree to which the machine representation of a number approximates an absolute definition of that number; accuracy is the degree to which calculated results approximate an absolute definition of the results. Thus while precision contributes to accuracy, it is not the only factor, and how calculations are produced may affect accuracy even more than precision. We have already confronted cases where trig function processing can cause problems (V2N2p1 and V2N5p4), and there are many cases where truncation at the LSD causes grief. Professor W Kahan at UC (Berkeley) examines machine accuracy versus apparent precision in a recent memo (of limited circulation), introducing some concepts which users may find helpful in the analysis of calculated results. He states that while the easiest way for a user to find out what's happening is via a so-called Forward Error

Analysis (FEA), what must usually be made is a Backwards Error Analysis (BEA). If a function f(x) is approximated with machine processing by F(x), then F(x) submits to a satisfactory FEA when F(x)=almost f(x). But this may only be true when the x specified in f(x) is exactly the same as the x used in F(x). What usually happens is that (x)=almost f(almost x) which can easily cause F(x) to be nowhere near f(x), and is a situation which requires BEA according to Kahan. Keeping in mind the 2 almosts and their possible cumulative effect should help the user understand what's going on, whether he approaches error analysis quantitatively or just qualitatively. Members are invited to further explore FEAs and BEAs vis-a-vis the PPCs.

Designing a Practical File Manager (59/PC-100A)

While it is tempting to try to devise ingenious search techniques (V3N2p3) to minimize the number of data registers required for arbitrary-key access, it may turn out that for most practical applications execution speed is more important than register economy.

The following program ties up all possible data registers, but is fast and easy to use. Members are invited to try it (or their own versions) in real-world applications and to report results, suggest improvements, etc. Files are organized by sets of 2 mag cards, each pair holding up to 99 files whose keys are 1 or 2-digit numbers in the 1-99 range. File contents (records) are interpreted as data when in the 10^{-99} to 10^{9} range, and as character strings when $\geq 10^{9}$. Provision is made to assign an identifying number to each card-pair, and this tag along with all stored records is preserved as a card-pair is read in from time to time for file reconfiguration. Files may be accessed in any order for store, recall, add, or subtract operations, and a listing made at any time of all non-zero records. A printed record is made of all transactions, and users reconfiguring files many times may wish to include revision information in a card-pair identifier. For example 25.12 might identify the 12th revision of card-set 25. This tag is mag-card-stored in Reg 0, but preserved in H4 during file reconfiguration. Reserve an otherwise blank card to serve as a master for transferring the program to side 1 of each new card-pair, keeping in mind that steps 160-239 are treated as data.

TI-59/PC-100A Program: File Manager Ed

User Instructions:

```
1.  To Prepare a File Manager Master Card:
   A. Write the program listed below into blank user memory.
   B. In RUN mode key 10 Op 17 1 2nd Write.
   C. Insert card, and record.
   D. Mark card:  "File Manager Master".
2.  To Prepare for Generating a New File Block:
   A. In RUN mode key 10 Op 17 CLR.
   B. Insert the File Manager Master Card, and read it.
   C. Key card number (any real), and press 2nd E'.
3.  File Processing:
   A. To Address a File:  Key its 1 or 2-digit key, press A, see its content
      displayed.
   B. To store a Record: Key the value (numbers ≥ 10^9 are interpreted as
      character strings). If 3a was just performed, press R/S. Else press SBR
      STO.
```

3. c. To recall a Record: Press SBR RCL.
   A. To Add to a Record: Key value, press SBR SUM.
   B. To Subtract from a Record: Key value, press +/- SBR SUM
   C. To List all non-zero Files: Press SBR List
   D. To Record Current File Configuration: Press E, of for new or revised card ID, key ID, press 2nd E'; record all 4 banks on 2 mag cards.
4. To Address an Old File Block:
   A. Read the 4 card-sides into memory, with a 159.99 partition.
   B. Press C and do step 3 as desired.

Program Listing:

```
000:  Lbl A STO0 Lbl D 261745 Op4 RCL0 Adv Op6 RCL*0 rtn Lbl STO STO*0
      363732 Lbl B Op04 1 EE 9
039:  x:t CLR RCL*0 x≥t 049 Op6 rtn Op2 Op5 rtn LR 351527 GTO B Lbl SUM
066:  x:t 364130 Op4 x:t SUM*0 Op6 rtn Lbl List Op00 27243637 Op2 Adv
096:  Op05 99 STO0 CP RCL*0 x=t 118 D 35153516 B Dsz0 102 Adv Adv Adv CLR
126:  R/S Lbl E' HIR4 31 Lbl E 153516 Op4 HIR14 STO0 Op6 GTO 122 Lbl C RCL0
      HIR4 GTO E
```

Advanced Programming Techniques V: Designing Operating Systems

Computer operating systems (OS) generally consist of a control or executive program, and the programs it directs to manage the required communication between mainframe computers, their peripheral devices, and users... utilities such as compilers, assemblers, loaders, editors, file maintainers, interrupt processors, job schedulers, job control language interpreters, etc. Among these utilities, one of the more challenging to design is a compiler, which translates a high order language (HOL) into assembly (AL), or machine language (ML). The difficulty lies mainly in the interface between how a human can best express what he wants a program to do, and how a machine can be made to implement his intent. Humans tend to think most effectively using concept-linked symbol strings, and HOLs (FORTRAN, BASIC, COBOL, APL, etc) are designed to let the programmer use familiar symbols as "naturally" as possible. An AL is almost understandable to the machine: each instruction is symbolic, but has a one-to-one correspondence with a ML instruction (a binary number). An assembler translates AL code into ML code, and a loader puts the ML code into the proper memory locations.

PPCese has some of the characteristics of HOLs, ALs, and MLs, but there is not much in common with any one of these. So it's a challenge even with the powerful TI-59/PC-100A combination to simulate even just a part of a mainframe compiler and its associated OS support. The program which follows simulates the implementation of a BASIC programming construct via a remote terminal. The user "types" an assignment statement consisting of single-character variables and operators, following the usual convention that only a single variable appears to the left of the = sign. The usual + - * / arithmetic and ^ exponent symbols are used, along with parenthetical nesting. Following input of up to 20 print codes (a variable may be any of the 59's 64 characters which is not one of the designated operators or parenthesis symbols), the program prints the statement as it would have been typed at a computer terminal, proceeds on into the interpretation process, calls for variable data to be input, runs the compiled/assembled/loaded code, and prints the answer.

The user may then make more runs with the same program with new input data, get a listing of the program, or begin compilation of a new statement.

Code transfer rules (V1N2p5 and V3N1p2) complicate the process of synthesizing instructions as data, and in order to dodge position B restrictions, AB positions are always set to 60 (the code for DEG, which does no harm). Positions CD are also set to 60 in cases where RCL n would otherwise be split by a 60 at the AB position. The resulting compiled code is perhaps somewhat realistically inefficient, just as real compilations are usually less efficient than human-designed AL code. AOS architecture essentially eliminates the real-computer requirement to translate HOL semantics into the proper ordering of machine instructions. Mechanizing this simulation on an RPN machine would be considerably more complex. There is program memory to space if some register assignments are changed, and members are invited to add enhancements, or mechanize better approaches.

TI-59/PC-100A Program: BASIC Operating System Simulator Ed

User Instructions:

1. **Initialize:** Press E
2. **Input BASIC Assignment Statement:** Key the 2-digit print code for a character representing either a variable or an operator, press R/S. Repeat for up to 20 characters. The first character must be a variable, the second the = symbol (64).
3. **Initiate processing:** Press A. (This step is done automatically following input of the 20th character). See the BASIC statement printed, followed a minute or so later by the cue: KEY n, where n is the character representing the first variable in the BASIC statement. Abort processing with R/S is the statement is incorrect as printed, and go to step 1.
4. **Run the Compiled Program:** Following each printed cue, key the data value for the indicated variable, press R/S. Following input of the last requested variable value, processing begins and the answer printed. For new inputs, press B, and repeat this step.
5. **To Get a Listing of the Compiled Code:** Press SBR List.
6. **For New Compilation:** Go to step 1.

Note: Record program with turn-on partition; program terminates with a 599.49 partition.

Program Listing:

```
000:  Lbl C' INV Stflg1 RCL*49 x:t 47 x=t 055 20 x=t 058 51 x=t 061 63 x=t
026:  064 60 x=t 067 55 x=t 070 56 x=t 073 x:t +/- STO*49 1 SUM48 RCL48
      Stflg1
054:  rtn 85 rtn 75 rtn 65 rtn 55 rtn 45 rtn 53 rtn 54 rtn Lbl E' C' Lbl B'
081:  Ifflg0 165 X RCL46 = SUM*47 .01 Prd46 1 SUM49 Ifflg1 110 Dsz45 109
107:  Stflg0 rtn INV Dsz45 135 RCL46 X 43 = SUM*47 .01 Prd46 Dsz45 134
132:  Stflg0 rtn 4 EE +/- 13 SUM*47 1 EE 37 Lbl A' Prd*47 1 SUM47 .01 STO46
158:  6 STO45 INV Stflg0 rtn Ifflg1 204 x:t 1 SUM49 x:t ÷ 10 = STO43 Int EE
182:  +/- 13 SUM*47 RCL43 INV Int X 100 + 7 = INV Log EE GTO 148 x:t 6 EE +/-
208:  12 SUM*47 1 EE 7 A' x:t GTO 081 Lbl E 6 Op17 CMs 0 STO0 Op00 20 STO49
235:  CLR R/S STO*49 Op20 Dsz49 236 Lbl A 1 STO46 20 STO49 RCL0 STO47 5 STO48
      CLR X
263:  100 + RCL*49 = INV Dsz49 293 INV Dsz47 293 Dsz48 262 = Op*46 1 SUM46
290:  GTO 258 = x:t RCL48 - 1 = X 2 = INV Log EE INV EE X x:t = Op*46 Op5
315:  Adv Adv Adv 1 INV SUM49 20 STO48 1 INV SUM0 .01 STO46 50 STO47 6 STO45
```

```
342:  RCL0 STO44 92 B' 95 B' E' Dsz44 352 INV Stflg1 x:t 6 x=t 369 1 EE 7
368:  A' 1.376901476 STO*47 Lbl B CLR 5 Op15 19 STO49 RCL0 STO44 RCL48 STO43
      Op00
402:  261745 Op1 CP 1 INV SUM49 RCL*49 CP x≥t 434 +/- Op2 Op5 R/S Prt STO*43
430:  1 INV SUM43 Dsz44 411 Op00 6400 + RCL20 = Op4 C Adv Op6 Adv Adv
456:  Adv R/S Lbl List Adv Op00 27243637 Op2 Op5 D Adv Adv Adv R/S

Note:  There are quite a few Dsz register and address operands which must be
synthesized.
```

Book Review: Programmable Calculators, R J and C J Sippl, 526 pages Matrix Publishers, 1978.

Charles Sippl (239) and his son Roger cover a lot of ground with this book, and held off publication many months so they could include the latest TI and HP machines. There is a lot of detail covering most (if not all) of the handheld programmables, as well as many of the modern desk-top machines, from hardware descriptions to elementary programming techniques. Considering the scope of this work and time pressures, perhaps the authors may be forgiven for some of the disorganization, repetition, and the errors and lack of clarity in some of the technical text. This work does expose the reader to important material not available elsewhere under one cover, and the serious PPC user will probably find enough helpful information in this book to make it a worthwhile buy, keeping in mind that he may want to consult other sources on important technical topics. And, it is certainly gratifying that Club coverage is both accurate and flattering!

Tips and Miscellany

More on Strange LRN Behavior (V3N1p5): Lou Cargile (625) reports having experienced a practical problem related to Jared's discovery, and notes 2 situations which can arise during program editing/debugging that can themselves create insidious new bugs: 1) When at step nnn in RUN mode, if you want to get to code headed by say Lbl Tan, it is easy to key GTO Ind by mistake, instead of GTO Tan. Assuming that you want to examine the code, the obvious next step is LRN, which reveals only that you are at step nnn+1, not the desired step, so you key LRN GTO Tan (properly), and proceed on without realizing that a code 22 had been written at step nnn; and 2) If in RUN mode you are single stepping through a sequence of the form: ...Dsz Ind nn N ... and stop after execution of the Ind and key LRN to see where you are, you will see the step containing N, not realizing that the nn in the previous step had been overwritten with code 11. In the first case, if you catch the GTO Ind mistake before pressing LRN, pressing another key (like CLR) first, prevents the unintentional overwrite. In the second case, about all I can suggest is to be wary of SSTing anywhere near an indirect Dsz, and if you stop to see where you are, follow the LRN with BST and verify the displayed code.

Local Club: Dave Johnston (5) and Maurice Swinnen (779) are forming a Washington DC area PPC users group. Contact either Dave or Maurice for more information.

Membership Address Changes: 343: 1325 Quaker St Golden CO 80401; 713: 804 Rhonda Dr Hephzibah, GA 30815; 836: 1539 Rainbow Ln Port Richey, FL 33568; 869: New York, not New Jersey.

Printer Spacing (58/59/PC-100A): Maurice Swinnen (779) has discovered that an Op0 Op5 line-space is 2 mm thinner than one produced by Adv. It turns out that an Op0 Op5 space causes the paper to be advanced exactly as far as occurs following character print, while Adv adds 2 mm. Obvious applications include printer graphics, and paper economy. Lou Cargile (625) found (independently) that Op0 Op5 helped to produce the print format he wanted for his Bridge Deal program (V3N2p5 steps 010-013).

A Tic-Tac-Toe Option (V2N10p6): Dave Leising (890) suggests keying GTO 227 R/S (SBR 227 saves a step) following printing of the first blank grid, if you want the machine to play first.

Basic Hardware Design Information: Dave suggests writing to the U S Patent Office for descriptions of the patents whose numbers are stamped on the back of your machine, as one way to learn more about hardware basics.

More on Fractured Display (V2N12p3): Kirk Gregg (748) found that at the time the display fractures, a special state prevails that causes the = or) functions to address Reg 0, i.e. Exc =, STO =, Prd =, etc execute as Exc 0, STO 0, Prd 0, etc. This special state prevails through the use of many built-in functions, but not after keying any of the numerals or CLR. The statistics, D.MS, and P-R functions do not work while this special state prevails. As Kirk notes, this phenomenon doesn't appear to have any useful applications, but a more thorough understanding of it might well lead to important discoveries. Incidently, Jared's fractured digits sequence (V3N1p1) produces the special Reg 0 state, but Fred's (V3N1p6) does not. Neither does Roger's out-of-bounds CROM call method (V3N3p2).

Use of Contributed Material: Maurice Swinnen (779) brings up the matter of member programs or routines submitted to me but which languish unpublished indefinitely, and for which there may be a desire to seek other vehicles for publication. Maurice suggests that a yes or no response from me upon receipt would be helpful, but I'd rather not have to commit myself too soon: topical interest fluctuates over periods of time, and I wouldn't want to reject material that might be just what I'm looking for later on. I suggest that in cases where you wish to publish elsewhere, if your program hasn't yet surfaced in 52-NOTES, send me a clear description if it, and a SASE if you would like it returned. If it is currently in the 52-NOTES publishing cycle, I will so inform you. If it has already appeared in 52-NOTES and you still wish to publish elsewhere, please cite the original 52-NOTES source to help minimize possible copyright contests. For the record: 52-NOTES continues to be published uncopyrighted.

Correction (V3N3p6): Mack Maloney reports that the TI DC 9105 adaptor produces calculator DC, not 120 VAC.

Used SR-52s: Members wishing to sell their SR-52s should contact Harold Bless (255), or a Mr S.Green at DAQ Electronics Lackland Dr Middlesex NJ (201) 560-0050.

Merged Code Labels (58/59): Jared Weinberger (221) points out that no mention has been made of the use of the 9 merged codes: 62,63 64,72,73,74,83,84 and 92 as labels. They all appear to work, but must be created synthetically, like pseudos or double-digit Dsz register addresses, and cannot be addressed from the keyboard.

Volume 3 Number 5 48/39/38 May 1978

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

An Analysis and Comparison of 3 Calendar Programs (59/PC-100A)

Both Jared Weinberger (221) and Lou Cargile (625) have written programs to generate and print calendars, using the ML-20 CROM program, and outputting in a standard month-by-month weekday column format. This is the sort of problem where neither its complexity nor the best approaches to its solution are apt to become fully apparent until at least one approach has been tried. And so it wasn't until I examined their programs in detail that I appreciated the degree of the challenge and was motivated to try to find a better approach. Both programs require 3 card-sides for recording, but neither uses all the recordably available memory, and since it is unlikely that a comparable program could be made to fit on a single mag card, the payoff in optimization is in the reduction of run time.

With the advantage of having Jared's and Lou's programs in hand I wrote a third one that runs in just three fourths the time required by either of the others (26 minutes versus 34-35 to print 12 months). The analysis and comparisons which follow are intended to be informative and constructive, and I hope it goes without saying that it is not my intention to impugn either Jared's or Lou's programming abilities. In the same spirit, I welcome better approaches from others who now have all 3 programs to start with, and it is my hope that we will all learn something from this exercise.

In addition to the usual user instructions and program listings, I am including high-level algorithms, which may make it easier to see how each program works. The most critical function to try to optimize is the generation of the print code required to print the successive integers 1,2,...31. Jared's approach to this requirement is the slowest, partly because he needed a routine to translate 4-digit year numbers besides the day numbers. Lou dodged this requirement by letting the printer print the year as the displayed value, with the month tag-printed in the same line via Op 4 Op 6. But I expect that a routine optimized to only translate integers in the 1-31 range into print code would still run slower than a print-code counter. Lou devised just such a counter, and this difference appears to give his program a slight speed advantage. But both Jared and Lou use a number of x-t comparisons where they could have used Dszs (which are faster) for branch control in their print code generators, and Lou uses 4 flags, 3 of which require repeated setting and resetting, as well as testing.

While inefficient approaches to meeting requirements posed only once per printed month aren't so critical as the once-per-day ones, the extra execution time can add up.

Both Jared and Lou constructed independent sequences to calculate the number of days in a month, since it apparently didn't occur to them that given the first day of 2 successive months, ML-20 returns the number of days in the first month. There is also some time lost putting month, day and year information into the required ML-20 input format, instead of directly storing the separate values in Reg 1, 2, and 3, and processing via SBR 086 or 177 (see V2N11p1 and Fred Fish's ML Survival Kit p 20-1).

So try all 3 programs, look them over, and see if you can write a faster one (one which runs say, under 2 minutes per month). Incidently, although the ML manual doesn't say so, 9999 appears to be the upper limit on the year, unless inputs are stored separately. But at some point the 366-day year every 4 years rule won't hold.

TI-59/PC-100A Program: Calendar Printer Jared Weinberger (221)

Algorithm:

1. Print the month header.
2. Call ML-20 D with 1st day of specified month and year to get day of week; convert the returned 1,2,...6,0 to 1,2,...7.
3. Determine number of days in month: For Feb: Call ML-20 A with Feb 28th, and ML-20 B with March 1st to get the number of days between Feb 28th and March 1st; use this result to get the number of days in Feb. For other months: Test for April, June, Sept, Nov to get the number of days.
4. Fill a cleared block of 7 registers with consecutive base 10 integers beginning with 1 in a position determined by the corresponding day of the week.
5. Convert the stored integers to print code via a translator; pack print buffers, and print a line of days.
6. Clear the integer block, fill with the next group of integers, and go to step 5.
7. Repeat steps 5 and 6 until all days have been printed.
8. For whole year: Increment month, and repeat steps 1-8 until all months have been printed.

User Instructions:

For one month: key month, press A; key year, press B; press D. For whole year: Key year, press B; press E.

Program Listing:

```
000:  Lbl A STO 29 rtn Lbl B STO30 rtn CP x=t 036 ÷ log Int STO8 Op28 INV log
      +
024:  x:t 100 Prd 21 8 x≥t 036 2 + 1 - Int SUM21 = X 10 Dsz8 023 CLR
050:  Exc21 x:t 1 x=t 059 x:t rtn 0 rtn RCL1 SBR 010 X 1 EE 6 = INV EE
073:  STO28 RCL2 SBR 010 + RCL28 = Op1 RCL3 SBR 010 X 1 EE 4 = INV EE STO28
      RCL4 ÷
103:  10 = Int SBR 010 + RCL28 = Op2 RCL4 ÷ 10 = INV Int X 10 = SBR 010 STO0
133:  RCL4 CP x=t 148 RCL0 INV x=t 148 + 1 = EE 8 STO28 RCL5 SBR 010 X 100 +
162:  RCL28 = Op3 RCL6 SBR 010 X 1 EE 6 = INV EE STO28 RCL7 SBR 010 + RCL28 =
      Op4
192:  Op5 rtn Lbl D Op0 RCL29 + 34 = STO28 RCL*28 Op1 RCL30 SBR 010 Op4 Op5
      Adv RCL31
223:  Op1 RCL32 Op2 RCL33 Op3 RCL34 Op4 Op5 Adv RCL29 X 2 INV log + R30 ÷ 4
      INV
252:  log + 1 = Pgm20 D CP INV x=t 265 7 STO10 RCL29 x:t 2 INV x=t 316 RCL30
277:  ÷ 4 INV log = STO0 + 228 = Pgm20 A RCL0 + 301 = Pgm 20 B Pgm20 C x:t
306:  1 x=t 311 + 28 GTO 340 RCL29 x:t 4 x=t 338 6 x=t 338 9 x=t 338 11
333:  x=t 338 1 + 30 + 1 = STO11 Pgm 1 SBR CLR 8 x:t 1 STO13 RCL13 STO*10 1
359:  SUM10 SUM13 RCL10 INV x=t 354 SBR 061 0 STO7 Pgm 1 SBR CLR 1 STO12
      RCL11
384:  x:t RCL13 x=t 413 8 x:t RCL13 STO*12 1 SUM12 SUM13 RCL12 INV x=t 382
      SBR
408:  061 GTO 372 SBR 061 Adv Adv rtn Lbl E 0 STO 29 1 SUM29 SBR 197 12 x:t
433:  RCL29 INV x=t 424 Adv Adv rtn
```

Prestored Data:

```
31:  3641003032 37410043 1700372300 2135003613 2513314000 2117144000
37:  3013351523 1333352427 3013450000 2541311700 2541274500 1341224000
43:  3617333740 3215374000 3132424000 1617154000
```

TI-59/PC-100A Program Calendar Printer — Lou Cargile (625)

Algorithm:

1. Determine whether to print 1 month or balance of specified year.
2. Call ML-20 D with 1st day of the specified month and year to get day of week for 1st day; convert returned 1,2,...6,0 to 1,2,...7.
3. Check for leap year, Feb, odd or even month, July, Aug, to get number of days per month.
4. Print month header.
5. Use the calculated day of week to determine where to begin filling print buffers from a quasi base seven counter; print filled buffers and refill from the counter until all days have been printed.
6. For more than one month: Repeat steps 2-5, incrementing the month until all months for the specified year have been printed.

User Instructions: For one month: Key MM.YYYY, press B; for balance of year: Key MM.YYYY, press A.

Program Listing:

```
000:  12 x:t RCL8 INV x≥t 010 R/S 1 SUM6 GTO 026 Lbl A INV Stflg4 INV Stflg
023:  0 STO6 RCL6 STO0 Int STO9 INV SUM0 X 100 + 1 = SUM0 RCL6 INV Int X
      10000 =
056:  STO10 RCL0 Pgm20 D ST07 0 x:t RCL7 INV x=t 076 7 SUM7 RCL10 ÷ 4 = INV
      Int
083:  INV x=t 089 Stflg0 31 STO13 0 x:t RCL8 ÷ 2 = INV Int INV x=t 134 R8
108:  x:t 2 INV x=t 129 29 STO13 INV Ifflg0 145 1 SUM13 GTO 145 7 x:t
131:  GTO 138 RCL8 x:t 7 INV x≥t 145 1 SUM13 RCL8 + 47 = STO9 Op0 RCL*9 Op4
159:  RCL10 Op6 SBR368 RCL46 Op1 R45 Op2 RCL44 Op3 RCL43 Op4 Op5 SBR369 INV
188:  Stflg1 INV Stflg3 1 ST00 0 STO14 STO16 5 STO15 INV Stflg2 RCL14 SBR289
      RCL14
214:  SUM16 SBR236 Ifflg2 204 SBR236 Ifflg2 204 SBR236 GTO 204 Dzz15
238:  275 RCL16 Int Op*0 Op20 RCL0 x:t 5 INV x=t 266 1 STO0 Op5 Stflg2 Ifflg3
264:  281 RCL16 INV Int STO16 5 STO15 100 Prd16 rtn Adv INV Ifflg4 288 R/S
288:  RST Dsz7 338 Ifflg1 304 .01 SUM14 Stflf1 .01 SUM14 INV Dsz13 332
314:  RCL14 INV Int x:t .08 x=t 339 .13 x=t 345 rtn 0 STO14 Stflg3 rtn rtn
339:  .02 SUM14 rtn .88 SUM14 RCL14 Int x:t 1 x=t 359 rtn SUM14 rtn Lbl B
364:  Stflg4 GTO 021 RCL47 Op1 Op2 Op3 Op4 Op5 rtn
```

Prestored Data:

```
43:  21000036 4300003700 370000 36000030 2020202020 25133100 21171400
50:  30133500 13333500 31034500 25413117 25412745 13412200 36173337
57:  32153700 31324200 16171500
```

TI-59/PC-100A Program: Calendar Printer — Ed

Algorithm:

1. Check for ML module presence; alert user with flashing display if absent.
2. Determine user's choice: 1 month, balance of start year, or print from start month and year until end of specified year.
3. Call ML-20 SBR 086 with 1st day of specified month and year to get corresponding absolute day, and ML-20 SBR 177 to get corresponding day of week. Call ML-20 SBR 086 with 1st day of next month and calculate difference between returned absolute day value, and first one to get number of days in specified month.
4. Print month headers.

5. Use calculated day of week to initialize a general purpose print-buffer-packer routine to print first line of days, followed by successive lines until all days have been printed. Use a quasi base seven counter to supply the print-buffer-packer routine with code for successive integers.
6. For more months, as determined by step 2, cycle months, and increment years as required; repeat steps 3-5 until done.

User Instructions:

1. Initialize: Press A, see module confirmation printed; flashed display indicated wrong module.
2. Key start year, press R/S, key start month, press R/S.
3. For 1 month: Press B; For balance of year: Press C; for specified period: Key end year, press D.

Program Listing:

```
000:  Lbl C' 5 STO17 1 EE 8 STO19 CLR RCL21 Op*20 0 STO21 1 SUM20 Dsz18 037 1
      STO20
028:  Op5 Op0 Stflg0 4 STO18 rtn Lbl E' Op28 INV Dsz11 053 INV Dsz12 066 rtn
053:  1 STO8 Op29 10 STO11 8 STO12 rtn 2 SUM08 9 STO12 rtn Lbl D' INV Stflg0
      CLR 1
080:  x:t E' RCL9 x=t 093 X RCL19 = SUM21 100 INV Prd19 INV Dsz17 145 RCL8 X
107:  RCL19 = SUM21 100 INV Prd19 Dsz15 128 0 STO18 C' Adv rtn INV Dsz17 149
133:  100 INV Prd19 Dsz17 D' C' GTO D' C' GTO 104 C' Ifflg0 D' GTO 133
156:  Lbl A' Op00 1 STO8 ST09 10 STO11 8 STO12 5 STO17 4 STO18 1 EE 8 STO19
      1 STO20 0 STO21
189:  RCL16 x:t 1 x=t 232 2 x=t 234 3 x=t 244 4 x=t 260 5 x=t 282 6 x=t
214:  274 2 STO17 1 STO18 100 STO19 4 STO20 D' rtn D' rtn 2 STO17 100 STO19
      D' rtn
244:  4 STO17 3 STO18 1 EE 6 STO19 2 STO20 D' rtn 1 STO17 3 STO18 1 STO19 2
      STO20 D'
273:  rtn 1 STO18 4 STO20 D' rtn 3 STO17 2 STO 18 1 EE 4 STO19 3 STO20 D' rtn
      Lbl A 1
301:  x:t Pgm1 SBR Write x=t 312 Op55 R/S Adv Adv Adv R/S STO13 R/S STO14
321:  R/S Lbl B RCL14 ST01 1 STO2 RCL13 STO3 Pgm20 SBR086 STO10 Pgm20 SBR177
      STO16 RCL14
351:  + 1 = STO1 RCL13 STO3 Pgm20 SBR086 - RCL10 = STO15 Op0 RCL14 + 47 =
      STO20 RCL*20
383:  Op4 RCL13 Op6 RCL47 Op1 RCL46 Op2 RCL45 Op3 RCL44 Op4 Op5 A' rtn Lbl C
      13 -
414:  RCL14 = STO22 B 1 SUM14 Dsz22 419 CLR rtn Lbl D - RCL13 + 1 = STO23 C 1
      SUM13
443:  STO14 Dsz23 439 Adv Adv R/S
```

Prestored Data:

```
44:  2135003613 1700372300 37410043 3641003032 251331 211714 301335
51:  133335 301345 25413117 25412745 134122 36173337 321537 313242
59:  161715
```

Program/Routine Copyright

Legal and legislative bodies are recognizing that it is difficult to apply copyright law (even the new one) to computer-readable works, so it is not surprising that we laymen are confused as to what is and isn't allowed. Ken Widelitz, a lawyer who writes the Legal/Business Forum in **Kilobaud** magazine, addresses this topic (Nov 77 p14 and Apr 78 p6) in some detail. An important concept which Widelitz touches on is the so-called Fair Use Doctrine, which effectively says that the less you copy, and the farther removed your use is from competing in the marketplace with the original, the surer you can be that you are acting within the law. An important widely accepted legal interpretation says that ideas themselves are not copyrightable; the expression of them is. Then there is the question of "derivative" works. Widelitz notes that there is no clear-cut dividing line between vary basic techniques (such a commonly used sorting routines) which are generally held to be uncopyrightable, and more specialized ones which are, and recommends that users seek legal advice for important specific cases. Interested members will find Widelitz' 2 articles both informative and thought provoking,

and may find helpful a new National Bureau of Standards publication (#500-17, $4.00 USGPO Wash DC 20402): **"Copyright In Computer-Readable Works: Policy Impacts of Technological Change"**.

Dave Johnston (5) and Lee Eastman (713) wonder if translations of programs for different machines are protected by original copyrights. I suggest applying the Fair Use Doctrine, and if it appears that your intentions might conflict significantly with the originator's interests, get legal advice. Members who have been involved in computer-readable copyright litigation are invited to share their experiences.

52-NOTES Subject Index

Many of you have suggested that it would be helpful to have a topical index, and I agree, except perhaps as to how it should be generated and maintained. To a large extent, the choice of alphabetically listed names to categorize published material is subjective, and I suspect that the best index is generated by the user himself, and maintained as a continuously updated card file. Getting started is a bit of a chore, but refamiliarizes the user with the published material, as well as providing him with a relevant reference. However, for those who don't want to bother, here is an index which I have incorporated into a new Club brochure to help prospective members assess 52-NOTES, and identify back issues of interest. It covers V1N1 through V3N4, with the V, N, and p designators omitted:

Periodical Review: TI-59 Newsletter, edited by Howard Gosman (754), monthly since Nov 77, $24 per year.

Howard and a few associates have launched this newsletter as a business venture (not a users club), claiming to target it to the non-mathematician, "average" PPC user. The content and style of the first 5 issues characterize this periodical as oriented toward the non-programmer interested in business, finance, and gambling. Except for a "beginners corner" which has addressed the use of built-in statistics functions, there is little in the way of how-to programming tutorials or other technical discussion. There is a fair amount of non-59 and even non-TI PPC coverage over a broad range of generally non-programming topics. Over 2000 subscribers are claimed, with growth to 5000 anticipated.

Tips and Miscellany

56/57/58 Program Exchange: Dave Johnston (5) has a new catalog dated 1 May 78 listing 32 math, 9 statistics, 10 operations research, 25 physics, 12 other physical sciences, 30 engineering and technology, 3 life and behavioral sciences, 13 games, 7 finance, and 1 general info-test routine... programs written for the SR-56, of which 47 have been translated for the 57, and 28 for the 58. 29 additional programs were written for only the 57; 32 for the 58. Send Dave 20¢ and a SASE for a copy of his catalog.

CROM Download into Used User Memory: A CROM download (Op 9) alters only the steps which it fills, leaving other steps as they were before the download. While the presence of user code remnants is unlikely to affect execution of the downloaded CROM code, it can be a source of confusion when the CROM code is listed. Fred Fitzgerald (252) suggested (later confirmed by Tom Cox) that just such a situation caused the Leisure Library-13 HIR confusion (V3N1p5, V3N2p6, and V3N3p6): The HIR instruction at step 240 was a user-code remnant, not part of the downloaded CROM. The safest way to avoid such confusion is to cycle the power switch before downloading.

TI Service Manuals and Parts: David Swindell (877) reports that detailed service manuals and some parts are available for the older TI machines from Bootstrap Enterprises, Box 64, Richardson, TX 75080.

More on Memory Add-On (V3N3p4): Simon Hughes (837) reports that he has successfully modified his 58, and has volunteered assistance to Toronto-area members who need help. Simon's business phone is (416) 294-9838.

EE and Eng (57,58,59): Fred Fish (606) notes that INV Eng returns either an EE or Eng display (hard or soft) to a turn-on display immediately, while INV EE on a soft EE display doesn't remove the EE format until another key is pressed. As the manual notes (V-8), the Eng format continues to prevail following INV EE or CLR on an Eng display.

CROM Use of Flags for Print Suppression: Roy Chardon (515) noticed that RE-2 uses flag 0 to suppress printing, but that no mention of this is made in the RE manual. Further investigation reveals that RE-2 never sets flag 0, but that RE-3 and RE-13 do, prior to calls to RE-2 to bypass a Prt which would otherwise execute with printer connection. Similarly, RE-4 uses flag 5 to suppress printing, but never sets it. RE-13 sets flag 5 in the same sequence in which it sets flag 0 on the call to RE-2, which in turn calls RE-4. Other members finding similar flag use in CROM code are invited to share their discoveries, to help minimize the unsuppressible printing problem, which is often a deterrent to CROM use by user programs.

Volume 3 Number 6 48/39/38 June 1978

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

Friendly Competition

It's a bit disappointing that none of HP's newly announced Series E calculators is even on a par with the 67. So there are still no HP PPCs qualified to match the TI-58 or 59 in our on-going Friendly Competition (V2N4) with HP users. However, continuing on with an old counter-challenge, Hal Brown (V2N8p3 and V2N12p5) has collaborated with another HP user in improving his matrix program (see **PPC Journal** (formerly **65-Notes**) V5N4P6,24) such that although manual rearrangements are still required in some cases, the new version runs faster, and can solve a system of 5 simultaneous equations as well as calculate the determinant and inverse of a 5 X 5 matrix. Avid SR-52 users may find it challenging to translate the Brown/Robinson program, and see if they can produce as good or better 52 versions.

But until HP (or some other PPC manufacturer) does catch up, 58/59 users will have to content themselves with same-machine challenges. The calendar one (V3N5) is turning into a lively contest, and it is now apparent that beating 2 minutes per printed month (V3N5p2) isn't too hard to do. Lou Cargile (625) and Fred Fitzgerald (252) are about even with programs which print a year in ten minutes. Lou uses indirect subroutine calls to specific entry points in long strings of numeral instructions to generate print-code strings in the display. Such calls are fast, but time is sometimes wasted getting to the next rtn after the display has been filled, and this approach requires specialized processing for non-31-day months. Fred takes a related, but somewhat different approach, using indirect subroutine calls to short sequences of the form: a.bc rtn, where a is the print-code for the tens place, and bc the print-code for the units place for all the days (1-31).

But it turns out that indirect recalls are faster than indirect subroutine calls to even the shortest sequence: rtn, and Maurice Swinnen (779) was able to devise a single-month printer which executes in 40-45 seconds using indirect recalls from a block of consecutive registers, each containing 2 print codes for 2 of the integers 1-31. Indidently, Maurice included a detailed algorithm along with mag cards, and listings, greatly facilitating the job of determining how his program works. While close to the fastest calendar printer I've yet seen, Maurice's program does a good deal of special-case handling, which offsets the speed gain of the print-code retrieval mechanization, and leaves no room for routines to handle more than one month at a time.

Bill Skillman (710) worked out a simpler way to pack print buffers for all cases from a block of consecutive registers, each containing print-code for a single day. His approach is the fastest yet: 7½ minutes per year. The program which follows is a revision of Bill's, cutting size and execution time a bit with the substitution of INV Dszs for x-t comparisons, and a bunch of equally spaced GTOs for a double indirect mechanization (the latter being more costly in registers, and slightly slower, even though more elegant).

Record banks 1-3 with turn-on partition.

TI-59/PC-100A Program: Calendar Printer Bill Skillman (710)/Ed

Program Listing:

```
000:  GTO 127 GTO 025 GTO 040 GTO 051 GTO 066 GTO 097 GTO 112 Lbl A' Op0
025:  RCL*2 Op22 EE 6 INV EE Op1 + INV Dsz4 1' RCL*2 Op22 = Op1 INV Dsz4 1'
051:  RCL*2 Op22 EE 4 INV EE Op2 + INV Dsz4 1' RCL*2 Op22 EE 2 +/- INV EE S5
077:  Int = Op2 RCL5 INV Int EE 10 INV EE Op3 + INV Dsz4 1' RCL*2 Op22
101:  EE 2 = INV EE Op3 INV Dsz4 1' RCL*2 Op22 EE 6 INV EE OP4 + INV Dsz4
126:  1' RCL*2 Op22 = INV EE OP 4 Op5 Dsz4 A' Adv CLR rtn Lbl 1' Op5 CLR
149:  Adv rtn Lbl B RCL1 + 64 = ST07 1 STO2 RCL9 STO3 Pgm20 SBR 086 STO0 ÷
      x:t 7 +/- =
180:  Int X 7 + x:t = X 3 = STO6 Op21 RCL9 STO3 Pgm20 SBR 086 - RCL0 = STO4
      Op0
210:  RCL*7 Op4 RCL9 Op6 RCL64 Op1 RCL63 Op2 RCL63 Op3 RCL61 Op4 Op5 Op00 20
      STO2
242:  GTO*6 Lbl A STO9 STO3 R/S S01 1 x:t Pgm1 SBR Write INV x=t Grad 8 Op17
265:  Adv Adv Adv rtn Lbl C 13 - RCL1 = STO8 B Dsz8 279 rtn Lbl D - RCL9 + 1
      =
293:  STO77 C Op29 1 STO1 Dsz77 295 Adv Adv R/S
```

Pre-stored Data:

```
30:  2 3 4 5 6 7 8 9 12 201 202 203 204 205 206 207 208 209 212 301
50:  302 303 304 305 306 307 308 309 312 401 402 2135003613 1700372300
63:  37410043 3641003032 251331 211714 301335 133335 301345 25413117
71:  25412745 134122 36173337 321537 313242 161715
```

User Instructions:

1. Key start year, press A.
2. Key start month, press R/S; if wrong module, display flashes. For 1 Month: Press B; for balance of year, press C; for specified interval, key end year, press D.

Incidently, as some of us found from frustrating experience, ML-20 sometimes alters input month or year. The routine which calculates the day of the week (D or SBR 177) unfortunately used Reg 1 for temporary storage, and when the input months are either January or February, steps 112-115 decrement the year.

Carl Seel (328) notes that *"...the 4 year leap year rule doesn't hold for centesimal years (1900, 2000, 2100, ...) which aren't evenly divisible by 400."* ML-20 accounts for this, but as Charles Kluepfel (757) notes, Lou Cargile's V3N5p3 program doesn't, since in neither uses ML-20 to calculate the number of days in February, nor accounts for the centesimal rule in any other way. Charles also notes: "As to the far future, there is question whether 4000 AD will be leap year or not, as further corrections may be necessary to the calendar." Members wishing to explore time keeping in greater detail are invited to read "Measuring Time" by T R Parkin in **Popular Computing** V6N6p7. In this first of a 2-part article, Parkin presents recent Naval Observatory figures defining the mean solar year as 365.2421946412 days on 1 Jan 1975 and *"...slowing down by .53 seconds for each century after Jan 1900".*

After printing V3N6p2, I received a calendar printing program from Panos Galidas (207) which used Maurice's doubled print-code method, but with a simpler way to handle all the special cases, and which prints a year in 5½ minutes. But as I examined Panos' approach (via a detailed algorithm, which like Maurice's, expedited comprehension), it appeared that further speed enhancements could be made, along with squeezing the code into 3 memory banks. It turned out that some of Bill Skillman's approach was applicable, and that using H5-8 in place of Op1-4 print buffer loading instructions, would speed up the required left-right number shifting. The program which follows prints a year in just under 5 minutes (1978: 4 min 53 seconds), and actually has more prestored data than is evident at first glance. These data are listed as steps 465-719 for input convenience. I suggest first writing into program memory all listed steps: 000-326 and 465-719 with a 1 Op 17 partition; then storing the Reg 62-78 constants with an 8 Op 17 partition, in order to avoid data shifts due to insertions or deletions; record banks 1-3 with a 6 Op 17 partition. During program execution there is an effective 327.78 partition. The V3N6p2 user instructions apply, except that there is no CROM module test.

TI-59/PC-100A Program: Calendar Printer Ed,207,221,252,625,710,779

Program Listing:

```
000:  30 STO0 Op27 GTO 115 31 STO0 Nop Nop GTO 061 30 STO0 Op27 GTO 061 31
029:  STO0 Nop Nop GTO 078 31 STO0 Nop Nop GTO 096 30 STO0 Op27 GTO 096 31
056:  STO0 GTO 115 RCL*0 INV Dsz7 139 HIR5 INV Dsz7 172 2 SUM0 RCL*0 + 99.99
086:  = INV Dsz7 145 HIR6 Op20 RCL*0 INV Int INV Dsz7 157 HIR7 INV Dsz7 172
112:  2 SUM0 RCL*0 INV Dsz7 169 HIR8 INV Dsz7 172 2 SUM0 Op5 Op0 GTO 061
139:  EE HIR5 GTO 172 + RCL62 - RCL62 = HIR6 GTO 172 + RCL62 - RCL62 = HIR7
      GTO 172
169:  EE HIR8 Op5 Op0 Adv CLR rtn Lbl B RCL1 + 66 = ST01 1 STO2 RCL9 STO3
      Pgm20
198:  SBR086 STO0 ÷ x:t 7 +/- = Int X 7 + x:t = X 9 = STO6 Op21 RCL9 STO3
Pgm20
227:  SBR086 - RCL0 = STO7 Op00 8 Op17 RCL*8 Op4 RCL9 Op06 7 Op17 R66 Op1 R65
258:  Op2 RCL64 Op3 RCL63 Op5 Op5 Op0 GTO*6 Lbl A STO9 STO3 R/S STO1 rtn Lbl
      C 13 -
289:  RCL1 = STO4 SBR181 Dsz4 294 rtn Lbl D – RCL 9 + 1 = STO5 SBR286 Op29 1
      STO1
320:  Dsz5 312 Adv Adv R/S

465:  0 0 0 CLR' Ind E' E' 0 CLR' Ind 0 E' Ind E' E' 0 E' Ind 0 CLR'
485:  p31 E' E' 0 CLR' p31 0 E' p31 E' E' 0 E' p31 0 0 p31 E' E' 0 0
506:  p31 0 Rad tan E' E' 0 Rad tan 0 Deg tan E' E' 0 Deg tan 0 Abs
525:  tan E' E' 0 Abs tan 0 Ind tan E' E' 0 Ind tan 0 tan tan E' E'
544:  0 tan tan 0 CLR' tan E' E' 0 CLR' tan 0 E' tan E' E' - E' tan 0
564:  CLR' p21 E' E' 0 CLR' p21 0 E' p21 E' E' 0 E' p21 0 0 p21 E' E'
584:  0 0 p21 0 Rad CLR' E' E' 0 Rad CLR' 0 Deg CLR' E' E' 0 Deg CLR'
603:  0 Abs CLR' E' E' 0 Abs CLR' 0 Ind CLR' E' E' 0 Ind CLR' 0 tan CLR'
622:  E' E' 0 tan CLR' 0 CLR' CLR' E' E' 0 CLR' CLR' 0 E' CLR' E' E'
640:  0 E' CLR' 0 CLR' 1 E' E' 0 CLR' 1 0 E' 1 E' E' 0 E' 1 0 0 1 E'
663:  E' 0 0 1 0 Rad 0 E' E' 0 Rad 0 0 Deg 0 E' E' 0 Deg 0 0 Abs 0 E'
687:  E' 0 Abs 0 0 Ind 0 E' E' 0 ind 0 0 tan 0 E' E' 0 tan 0 0 CLR' 0
710:  E' E' 0 CLR' 0 0 0 0 E' E'
```

Prestored Data:

```
62:  1000 2135003613 1700372300 37410043 3641003032 251331 211714
69:  301335 133335 301345 25413117 25412745 134122 36173337 321537
77:  313242 161715
```

For those of us actively involved in the calendar printer competition, it has been (and may continue to be: I just thought of a way to save 24 steps and a little time in the V3N6p3 equally-spaced GTOs... and who knows, maybe there is a 4 min/year program on someone's drawing board...) a time-consuming, but stimulating and rewarding contest. To the few who already knew the important programming tools, techniques, and approaches illustrated by 52-NOTES calendar coverage, I apologize for wasting the space. But for most, taking the time and devoting the patience to finding out how and why each of the listed calendar proprams works, and why some are faster than others, will result in significantly broadening programming skills, and I hope, justify the 52-NOTES coverage. I expect to extend the range of future Friendly Competition topics, but there aren't many non-trivial problems whose solutions are as universally understood as optimizing a PPC calendar printer.

Book Review: Programmable Calculators - Business Applications; by Aronofsky, Frame and Greynolds; 203 pp, $8.98, McGraw-Hill 1978.

Tracy Bollinger (864) and James Merrill (693) brought this new book to my attention, and both report in essence that it would be most useful to those novice users who need more spoon-feeding than the users manuals provide, but who don't care to go beyond the manuals' programming scope. The title might be more appropriately worded: *"Learning How to Program Your TI-58/59/PC-100A For Elementary Business Applications"* since non-TI machines are not covered at all, and non 58/59 very little. The authors teach at Southern Methodist University, and write in an easy-to-comprehend textbook style, covering manual as well as basic programming topics. While many functions and concepts are more fully explained than in the 58/59 users manual, there are misleading oversimplifications, and a few technical errors. For example, Sec 8-2 titled: Indirect Instructions covers only indirect stores and recalls (although the text claims to cover indirect summing as well); page 143 confuses card protection with program protection, and page 172 incorrectly states that a programmed CP resets all flags. Important (to many of us) unannounced features such as the HIR operations and 2-digit Dsz registers are not mentioned at all.

Tips and Miscellany

Membership Address Changes: 185: 856th ASA Co APO NY, 09039; 246: 2554 Key Largo Ln Ft Lauderdale, FL 33312; 491: 10571 Kerrigan Ct Santee, CA 92071; 493: 2800 S Lamb Blvd #181 Las Vegas, NV 89121; 581: 10172 Saluda Ave San Diego, CA 92126; 663: change 8333 to 7830; 684: 1505 Locust St Pasadena, CA 91106; 755: 1046 Gen Allen Ln West Chester, PA 19380; 845: 10409 Towlston Rd Fairfax, VA 22030; 997: M E Holthaus 1181 E Walmut Bldg 5 #6 Carbondale, IL 62901.

CROM RST Behavior: Fred Fitzgerald (252) found a code 81 at ML-19 step 377, and reports that it executes as RST R/S (which I confirm). Indidently, Fred has made copies of CROM program listings arranged on 8½ X 11 sheets for modules 1 and 7 and will provide these to members, at $3.00 for one, $4.00 for both. See V2N5p4 for Fred's current address.

Structural and Applied Mechanics PPC Applications: Bob Thacker (30) has volunteered as a focal point for interested MEs, and wonders if anyone has tried clock speed-up mods for the 59.

Last-Step Instruction Execution under Programmed List (58/59/PC): Jared Weinberger (221) has found that the execution of a last-step rtn during a programmed List (V2N10p3) applies also to other last-step instructions. A sequence of the form: ***Lbl A*** *B seq1 rtn* ***Lbl B*** *List seq2 f seq3 rtn*, where f is an instruction located in the last step of the current partition, on a call to A lists seq2 and instruction f, and executes instruction f. If f is rtn, seq1 is executed following the listing of seq2 f; if f is R/S, execution halts at the last step; if f is RST, execution continues at step 000; but if f is any other instruction, an error condition is set and the program halts following its execution. Composite instructions beginning with f are completable manually from the keyboard. If f is C, and if routine C repartitions memory such that seq3 is executable, a call to A lists seq2 C, executes C then seq3, seq1. Members are invited to find new practical applications for any of this behavior.

Subroutine Execution from the Printer (52,56,58,59,PC): Jared found that with one of the new machines connected to the printer, a manually keyed SBR PRINT or SBR ADV will initiate processing at defined Prt or Adv labels, where the PRINT or ADV are the printer keys. With the 52, SBR PRINT or SBR ADV execute as GTO Prt or GTO Adv, as might be expected; with the 56, error states are produced. While this behavior may have little practical value, it demonstrated another backward link from the printer to the PPC, akin to PRINT or ADV writing code into PPC memory when in LRN mode (V1N3p6).

An All-Digits Generator: George Hartwig (638) suggests that 80 ÷ 81 = is a short way to produce .987654321. However, the 13-digit machines actually produce .9876543209876 and the 11-place 57 produces .98765432098. So depending upon use, it might be well to follow the 80÷81= with EE INV EE.

Strange Pause Effects (58/59): George found that if while holding down the GTO key during program execution to pause-display the results of each step, you key other keys, strange things happen in the display. Results appear to be digit-value and number-size dependent.

A Search for Useful CROM Dsz Loops (58/59): For routines which loop many times, significant execution time can be saved if the required code can be found in a CROM sequence (see V2N12p2). For example a sequence of the form: ***Lbl A*** *RC*n x=t C Dszn A ...****Lbl C****...* will perform sequential searches through a specified block of registers. But to do this acceptably for an unintended application, Lbl C needs to be followed by a rtn, with intervening steps that do no harm, and don't waste too much time. Members are invited to search CROM listings for such a sequence, or other potentially useful loops which could save time.

Shortened Paper Drive (PC-100/100A): Fred Fish (606) notes that the SR-60 manual advises users to precede the first print command with Adv to preclude reduced paper movement caused by tear-off tension through the drive mechanism, and that such advice applies also to the PC-100 and 100A. I find that the problem is minimized when the tear-angle is minimized: pull the paper down across the teeth, not up.

More on Runaway Motor (V3N3p5): S.H. Hartman (920) reports that the card-drive motor can (always?) be stopped by keying INV Write.

Linear Programming: John Hirsh (736) brings up this topic, and solicits membership interest vis-a-vis PPC applications.

EE on a 0-digit Mantissa (58/59): James Merrill (693) notes that the manually keyed sequence: 777777777 EE 34 = displays as 7.7777778D09, but as 7777777773 when followed by INV EE. They both represent the same number in the display register: 7.777777773000D09 which indicates that the 3 following the EE got appended to the man- tissa. In general, a manually keyed (or programmed and run) sequence of the form: abcdefghi EE j produces the integer: abcdefghij. Apparently the EE is ignored as a decapower prefix when the mantissa is too large for a turn-on display, but small enough to accept another digit, if such is keyed following the EE.

A Manual SBR Quirk (58/59): Izzy Nelken (576) found that a manually keyed SBR 0 x^2 in RUN mode squares the display, then executes code starting at step 000. This generalizes to the following: SBR ss m executes monadic function m on the display, then code starting at step 0ss; and SBR ss d executes dyadic function d on the display and on a second operand produced by the code starting at step 0ss.

Computer Listings of SR-52 Programs: Roger Cowell (1010) has written a computer program in HP-2000E BASIC which he runs from a highschool interactive terminal to print 58/59/PC-type listings from 2-digit SR-52 instruction code inputs. Write Roger at 115 Fairview Cir Webster, NY 14580 for details.

CROM Partitioning: Arthur Ehrlich (969) reports that steps 164-166 of LE-13 are 6 Op 17, which when executed in a 58, lock out all of user memory. So don't try to call that part of LE-13 from a user program in a 58!

ML-24/25 Precision: John Mickelsen (990) reports that the statute to nautical miles conversion factor is inaccurate (good to only 8 places). Users of these programs should check the values listed on pages 84 and 86 of the ML Manual against the accepted standards, and note to how many places they are good.

Card-Drive Adjustments (59): Some hardware-knowledgeable users have attempted to reduce card read/write errors by adjusting a motor speed pot. But Robert Cruse (889) quotes a TI source as saying that *"...alignment of the four heads and the azimuth of each..."* is more critical then speed adjustment. Members are reminded that user-performed hardware adjustments/mods may invalidate warrantees. Robert also reports the existence of *"...a new book on the repair and maintenance of the TI-59 for $11.00"*, but did not disclose the source or availability.

CLR for INV EE: Lou Cargile (625) and Fred Fish (606) suggest waiting until a CLR can be used safely to return the display to an INV EE format, saving a step.

Routines

CROM-Call Selective Trace (58/59/PC): For CROM programs such as ML-8 which call user-written routines, Jared Weinberger (221) suggests: ***Lbl A'*** *Ifflg0 B' Stflg9 B' INV Stflg9 Stflg0 rtn* ***Lbl B'****...rtn* as a preprocessor to the called routine (in this case A') which causes trace-printing of the routine itself (B') the first time it is called, but not thereafter.

A General Purpose Integer Routine (V2N12p1**):** George Rapasy (981) devised the sequence: *...CP x≥t 1' (CE - 1)* ***Lbl 1'*** *Int...* which is 3 steps shorter than Joel's, protects pending arithmetic, and can be made to work on all the TI PPCs except the 52.

Volume 3 Number 7 48/39/38 July 1978

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

Machine Number Scaling Terminology and Display Format Notation

One of the consequences of writing in a terse style is the communication breakdown caused by the misuse of key words. This happened in my discussion of HIR arithmetic (V2N9p2), and I will attempt here to summarize current usage, to propose better terminology for use with the PPCs, and to clarify what I meant to say.

The confusion stems from my use of the computerese terms: fixed-point and floating-point to describe machine-displayed number-representations, when these terms should be applied only to number representations actually used during number manipulations by the machine. In computerese, fixed-point arithmetic means that positioning of a radix point is up to the user: The machine performs arithmetic without scaling, sort of the way a sliderule does, and it might be helpful to think of fixed-point as meaning no-point. Early computers performed only fixed-point arithmetic via their microcode (the hardwired mechanism for executing user-written raw or assembled machine code). For example, an 8-bit binary machine performing unsigned fixed-point arithmetic would treat all numbers a integers on the 0 to 2^8-1 (0-255) range. Scaling to other ranges would have to be done manually, or written into the user's program, with the constraint that the difference between the largest and smallest numbers be less than 256. If the 52, 56, 58, 59 PPCs had been designed to perform fixed-point unsigned binary arithmetic with their existing register/memory capacities, they would treat all numbers as integers in the 0-2^{64}-1 (0-18446744073709551615) range.

Floating-point arithmetic requires number-representation which includes a scale factor, usually a power to which the machine's number base is raised. The PPCs actually perform binary-converted-to-decimal (BCD) floating-point arithmetic, which produces a decimal display faster than translated binary floating point arithmetic, but which requires greater memory capacity for the same precision. Of the available 64 bits, 52 represent the mantissa, 8 the scaling factor (decapower), and 4 the 2 signs (see V1N1p5).

While most modern general-purpose computers provide the user with separate machine instructions for either fixed-point or floating-point arithmetic, the PPCs provide the user with floating-point arithmetic only, giving him a choice of display. and herein lies a source of confusion. To make matters worse, certain display formats alter the value of the floating-point number in the display register, which is always in the BCD floating-point described in V1N1p5. So what we need are some handy, unambiguous

terms to fully describe the display format, keeping in mind that the internal representation of all numbers is always floating point. I propose that a "turn-on" display means the standard INV Fix display that prevails at machine turn-on; turn-on fix n: the standard display rounded to n places; turn-on EE: the fullest possible mantissa in scientific notation; fix n EE: mantissa rounded to n places; turn-on Eng: the fullest possible mantissa in engineering notation; and fix n Eng: mantissa rounded to n places. Let number type refer to mathematical definitions such as integer, fraction, real, complex, etc. So back to V2N9p2: Change full integer-fraction to real; floating point to turn-on or fix n EE, turn-on or fix n Eng; and fixed point fraction to turn-on or turn-on fix n fraction.

The "floating decimal point" referred to in the 58/59 manual on page II-8 means that the machine positions the displayed decimal point on accordance with the scaling information contained in the display register. Since they've probably been around too long to change, we'll just have to live with them, but the terms fixed-point and floating-point can be misleading even in the internal-to-the machine context. Fixed-point, as we have seen, really means no point, and a displayed or printed floating point number in an EE format has a fixed decimal point! For a detailed technical discussion of machine-representations and manipulation of numbers, see Chapter 4 (Vol 2) of Knuth's **"The Art of Computer Programming"**.

Special-case Search Programs

There is a set of problems which can be expressed generally as $f(a_1,a_2,...a_n) = g(a_1,a_2,...a_m)$, for which given different functions f and g, it is desired to determine what values of $a_1,a_2,...$ produce solutions. Quite often, there are no known analytic approaches, and solutions depend heavily on trial and error methods. One such problem was brought to my attention by Bob Anderson (506), and was posed as a challenge to computer hobbyists by **Kilobaud** (Dec 77 p 20 and 26; Apr 78 pp10-12). The requirement was to *"Write a program to find all 3-digit numbers for which the sum of the cube of the digits is equal to the number"*, or in mathese: $a^3+b^3+c^3= 100a + 10b + c$; a,b,c in the 0-9 range. There are several straight forward ways to attack the problem, but most are overly time-consuming: One Kilobaud reader submitted a TI-58 program which took 1½ hours to run! Bob has one (which I have not seen) that takes 28 minutes, and he thinks he's close to getting the bugs out of an 11½ minute program. Members are invited to mechanize solutions on any of the TI PPCs, and to send me their fastest. Valid algorithms will assume nothing a priori concerning the solution set: 000, 001, 153, 370, 371, and 407, although the trivial solutions 000 and 001 may be ignored. Any approach may be used which assures finding all solutions in the 002-960 range. Programs written for printerless operation should pause-display each solution, and will be given a 1½ second handicap; programs with R/S or HLT displayed solutions will have to be timed in segments. The program which follows runs in 17 min 53 sec with a call to A, and shouldn't be too hard to beat. For more than one run, registers 11-14 must be reinitialized. Replace the Prt at step 040 with Pause or R/S for printerless operation.

TI-58/59 (PC) Program: Solutions to $a^3+b^3+c^3=100a+10b+c$ Ed

Listing:

```
000:  Lbl A RCL*12 + RCL*13 + RCL*14 = x:t RCL 11 x=t 040 Dsz11 022 R/S Dsz14
      A
025:  10 STO 14 Dsz13 A 10 STO 13 Dsz12 A R/S Prt GTO 017
```

Prestored Data:

```
01:  0 1 8 27 64 125 216 343 512 729 960 10 7 1
```

This exercise suggests extending the problem to the more general specification: $a1^n+a2^n+...\ an^n=10^{n-1}a1+10^{n-2}a2+...\ an$, although even the fastest computers would be too slow for very large n, unless sig- nificant analytic shortcuts are available. Incidently, the April **Kilobaud** article does suggest some shortcuts, but the required special-case testing might well turn them into longcuts for PPCs. It's interesting that if the problem is further generalized to: $a_1^n+a_2^n+...\ a_n^n = b^{n-1}a_1+b^{n-2}a_2+...$ an where b is any number base, that for b=2, there are only the trivial solutions n=1, and a_i=0 for all i, or all but i=n. Perhaps it can be proven that there are no non-trivial solutions if b is not an integer. Members with number theory expertise are invited to comment, and to suggest other special-case search problems.

Code Synthesis via Op 8 Label Listing (58/59/PC)

While designing the BASIC Operating System Simulator (V3N4p4), Ibriefly considered synthesizing the object code with labels attached to each instruction to allow the code to be stored in data registers without the transfer restrictions, yet making it possible to list it "cleanly" via the Op 8 function. But I dropped the idea when it appeared that there would be no way to define register operands or other numerals. However, Tom Cox (9) has found that the Op 8 function recognizes the numerals 0-9 as labels, even though the branch instructions do not. So, putting to use Tom's and A B Winston's (V2N10p3) discoveries, and the fact that Op 8 will list repeated labels, one should be able to synthesize any code sequence for listing (but not execution). The only required non-standard conventions are that double digit register operands list sequentially, and that step numbers should be ignored. The following routine illustrates what can be done, and lists a Fibonacci Number generator routine written for the TI-57, when run with a call to E on a 58/59/PC combination*: **Lbl E** 10 Op 17 CMs 92 STO 90 76760869 STO 99 85760576 STO 98 32766676 STO 97 1005766176 STO96 9 Op 17 SBR 160 Adv Adv Adv R/S.* If actually used with a 57, only the printed nemonics apply: **LBL 5** + PAU x:t GTO 5.

Calendar Printing Competition (continued)

Tom Ferguson (421) noted that I had omitted the prime symbol for the CLR' at step 713 of the V3N6p3 program. I tried running the program as printed, and it appeared to work. However, as Robert Petrie (632) found, if the first day of a month falls on Thursday, the 1 won't print. It turns out that the 5 of the CLR's code 25 multiplies the print code in Reg 30 by 10^{50}, leaving it with no fractional part for the INV Int shift at steps 098-099.

It has been interesting to watch the calendar printer programs get faster and faster, and rather amazing that the best current programs run in less than a tenth the time required by the first. But perhaps even more interesting is the trend of programmer motivation during this time. The 26-minute V3N5p4 challenger was fairly easy to beat, and quite a few members were quick to respond. But as the time got down to 10 minutes and less, what is sometimes called the Existence Theorem began to become an influence: if you doubt that a faster program can be written, you won't bother trying

to write it. And I expect some of us may have been influenced by a sense of lily-gilding: Why bother trimming a second or two if the program is already (or presumed to be) far in the lead. Well, fortunately at least 2 members have ignored (or successfully overcome) these deterrents, and have continued to press on with faster programs. As late as the last week of June, Lou Cargile and Panos Galidas were just about even, with Lou barely edging out Panos 3 min 15 sec to 3 min 17 sec. At this point, both were using just about all the available memory, putting as much code in-line (which is faster than subroutine calls, or looping) as they could, and devising a few new special-case processing tricks. HIR print-code processing is still de rigueur, but now (Lou's and Panos' latest both arrived in the 5 July mail) Lou has taken a significant lead by combining carefully chosen integer/fractions, which help to speed things up sufficiently to get a year (1978) printed in 2 min 57 sec! Panos hasn't slackened off in his efforts, but his latest gained him only 2 seconds, matching Lou's earlier 3 min 15 sec program.

Intense as the competition has been, all concerned have so far conducted themselves as gentlepersons, and at least one has managed to maintain a good sense of humor: Maurice Swinnen sent in a tongue-in-cheek "winner" which tells the user in 8 seconds which of 14 preprinted year-sets to use for a given year! It's actually quite practical to use, and might be an interesting challenge to 56/57 users. Here's the algorithm: From an input year, calculate the day of the week for Jan 1st. and use this to identify one of 7 pre-printed year-sets for non-leap years. For leap years, add 7 to the calculated year day to identify one of 7 additional year-sets, each having a 29-day February.

I'm listing Lou's 2 min 57 sec program with all prestored data as data to make it easier to see how it works. 13-digit numbers may be synthesized by first storing the 9 or 10 LSDs appropriately scaled such that when the remaining MSDs are summed, the LSD occupies the 13th mantissa position. For example, to store: 2.000311000312, key 311000312 ÷ 1 EE 12 = STO n 2 SUM n; to store: .2800021171400, key 21171400 ÷ 1 EE 13 = STO n .28 SUM n.

TI-59/PC Program: Calendar Printer — Lou Cargile (625)

User Instructions:

Key month, press A; key year press R/S. For one month, press B; for balance of year press C; for specified interval, key end year, press D.

Listing:

```
000:  Lbl A STO1 STO4 STO5 - 13 = +/- STO 47 8 Op17 67 SUM04 2 STO45 RCL5 R/S
      HIR4 R/S
031:  Lbl B 0 STO47 Lbl C HIR14 STO3 Pgm20 SBR086 Pgm20 SBR177 GTO 077 7 x:t
      RCL1
059:  INV x≥t 066 - 7 = STO1 +/- ST00 12 SUM0 CP RCL*4 HIR8 Int INV x=t 125
085:  3 STO45 HIR14 ÷ 4 = INV Int INV x=t 124 HIR14 ÷ 400 = INV Int x=t 123
111:  HIR14 ÷ 100 = INV Int x=t 124 1. SUM1 - 15 = +/- ST04 4 Prd3 STO44
HIR14
141:  Op6 RCL*3 STO42 Op23 RCL*3 STO41 RCL64 Op1 RCL65 Op2 RCL66 Op3 RCL67
      Op4 Op5
171:  INV Dsz45 188 Op23 RCL*3 STO40 Op23 RCL*3 STO39 RCL*0 SUM0 HIR5 RCL*0 +
       99 =
200:  HIR6 Op20 RCL*0 SUM0 - 1.99 = H7 RCL*0 SUM0 H8 Op5 Dsz44 188 Op0 RCL*0
232:  SUM0 HIR5 RCL*0 + 99 = H6 Op20 RCL*0 x=t 278 SUM0 - 1.99 = HIR7 RCL*0
      SUM0
265:  HIR8 Op5 Op0 RCL*0 x=t 280 HIR5 Op5 Adv Op25 Op24 Dsz47 055 CLR rtn
291:  Lbl D STO46 C Adv 12 INV SUM4 STO 47 1 HIR34 STO5 STO1 HIR14 x:t RCL46
      x≥t 295 R/S
```

Prestored Data:

```
02:  7 0 0 0 2 2 2 2 2 2.000000000002 2.000002000003 2.000003000004
14:  2.000004000005 2.000005000006 2.000006000007 2.000007000010
```

```
18:  2.000010000011 2.000011000012 2.000012000201 2.000201000202
22:  2.000202000203 2.000203000204 2.000204000205 2.000205000206
26:  2.000206000207 2.000207000210 2.000210000211 2.000211000212
30:  2.000212000301 2.000301000302 2.000302000303 2.000303000304
34:  2.000304000305 2.000305000306 2.000306000307 2.000307000310
38:  2.000310000311 0 0 0 0 0 0 0 0 2.000402 2.000401000402
50:  2.000312000401 2.000311000312 2.000401 2.000312000401
55:  2.000311000312 0 0 2.000312 2.000311000312 0 0 0 2.000311
64:  3641003032 37410043 1700372300 2135003613 3.0000251331 .28000211714
70:  3.0000301335 2.0000133335 3.0000301345 2.000025413117 3.000025412745
75:  3.0000134122 2.000036173337 3.0000321537 2.0000313242 3.0000161715
```

Routines

More on the Mathematical Integer Function (V3N6p6): Joel Pitcairn (514) and Charles Kluepfel (757) note that George's routine won't handle negative integers properly. The following appears to work, and is the same length as Joel's (each would be one step longer if () are used to preserve the arithmetic stack): *...CP x≥t 1' + INV Int x=t 1' 1 +/- -* ***Lbl 1'*** *= Int...* . Joel's and Charles' findings point out the importance of thorough testing to validate new routines, and the danger of neglecting to try what may seem to be trivial types of inputs.

A Self-calling Subroutine (V3N2p1**):** R.G. Snow (212) suggests a sequence which generalizes to the form*:* ***Lbl A*** *seq1 B seq2 A seq3 R/S* ***Lbl B*** *seq4 rtn* to do the following: 6(seq1 seq4 seq2) then seq1 seq4 seq3. Here, as R.G. notes, after all 6 subroutine levels have been used, the rtn transfers execution to the beginning of seq3 instead of seq2. However, when this routine stops at the R/S, there are 5 pending returns to the beginning of seq3, as may be demonstrated by replacing the R/S with rtn, and running: ***Lbl A*** *1 Prt B 2 Prt A 3 Prt rtn* ***LBl*** *4 Prt rtn* with the printer connected.

A Short INV Int X 100 = Replacement (58/59): In cases where you need to save steps, but don't mind extending execution time, R.G. Snow and Lee Eastman (713) suggest: D.MS H18. Incidently, some of the results appearing in H1, H2, and H8 following D.MS and INV D.MS are difficult to predict. For example, D.MS appears to return in H1: 36 X Int + 60 X .ff + 100 X .00ff... of the input number, but does it always? And, what is the formula for H2 and H8 contents following INV D.MS?

Print Code References (58/59/PC): R.G. suggests: ***Lbl A*** *8 STO 0 STO 1 CLR* ***Lbl A'*** *+ Op4 Op6 1 Dsz0 A' 8 STO 00 3 Dsz1 A' CLR R/S* if you don't have your owner's manual handy, and need a quick printcode reference. Run with a call to A. If you want to know what character any given 2-digit number will produce, just key the number and Op4 Op06 in RUN mode.

Improved Open Parentheses Counter (58/59): Jared Weinberger (221) has devised a version of his V1N7p5 routine for the new machines, which returns with the original number of open parentheses prevailing at the time of call maintained. The count of this number is in Reg 1: ***Lbl A*** *x:t 10 STO 1* ***Lbl 1'*** *(INV Stflg 7 Op31 Op22 Op18 Ifflg 7 1' CE* ***Lbl 2'*** *Dsz2 3' x:t rtn* ***Lbl 3'*** *) GTO 2'.*

Selective Register Listing (58/59/PC): Here is another routine from Jared, which printer-lists 10-register blocks specified by an input a.bb where Reg a_0 is the first and Reg bb_0-1 the last: ***Lbl A*** *(x:t Op16 INV Int X 10 + 1) ((x:t X (INV Int X 100) Op17 1) Int X 10) INV Lst Op17 rtn.* For example, a call to A with 2.05 displayed prints the contents of Reg 20-49.

Hyperbolic Trig Functions Shortcut (V2N7p3**):** John Van Wye (982) notes that the Sinclair approach becomes increasingly inaccurate as the input argument increases. It turns out that a slight improvement results when the machine is put into radian mode. For 58/59 users, Fred Fish's V2N11p1 approach is both shorter and more accurate, for all inputs.

A Shell-Sort Application (V3N2p5**):** Robert Trost (996) has found this routine helpful *"...as a preparation for the Histogram Construction Program"* (CROM ST-09). As Robert implies, the sorter would be more useful applied to a large number of inputs than for ordering output cells, since the max number of calls for ST-09 is 12.

Tips and Miscellany

Detecting Machine Hardware Changes (V3N4p1**):** Robert Petrie (632) has compared two 59s of different vintage (24-77 and 04-78) and notes that the added discrete components are of tighter tolerance, the new card reader has a thin Teflon sheet between the pressure pad and the read head, and a protective sheet of Mylar has been placed between the keyboard buttons and their contact surfaces. from what Robert and several other members report, old machines reconditioned by TI have all the new features, and are better than new ones of the old design.

SR-56 Pseudos: David Swindell (877) reports that a hardware jump between two specified points on the 56 PC board makes it possible to create pseudos. From what David describes, some of these are similar in behavior to the other PPC pseudos. Write David for details, bearing in mind that TI removes non-TI performed hardware mods when repairing machines, and that the mods themselves may void unexpired warrantees.

Printer-Connection Slowdown (58/59/PC): Rusty Wright (581) notes that programs run slower with printer connection than without. I find that a basic Dsz loop is about 3% slower, but that there is no measurable difference with 52 or 56 connection.

A Totally Blank Display (58/59): Izzy Nelken (576) was experimenting with sequences of the form: Pgm 11 SBR 999 R/S LRN (V3N3p2) and found that he could produce a totally blank display with (at turnon): LRN list 0 0 0 0 0 0 LRN Pgm 11 SBR 999 R/S LRN 5 (or any other numeral). This behavior appears to be consistent with the rules outlined in V3N3p2, and suggests a handy way to save battery power with a dormant turned-on machine. The required sequence appears to generalize to (in turn-on RUN mode): GTO 007 Pgm N SBR M R/S LRN where N is in the 10-17 range, and M is a positive integer which exceeds the number of steps in Pgm N.

Club Support of TI PPCs: Izzy asks if *"...the Club supports TI-55 and MBA programs"*. Subject matter concerning any TI PPC is considered for 52-NOTES coverage, but priority continues to be given to important discoveries, useful tips, clever routines, and illustrative programs.

56/57/58 Program Exchange: Exchange coordinator Dave Johnston (5) has moved to 11 Pine St Concord, NH 03301.

PC Carrying Case: Bill Fagerstrom (692) reports that Radio Shack carries a padded instrument briefcase designed for CB equipment, but in which a PC-100 will fit. Outside dimensions are 15x12x5 inches, and the case sells for $15.

Volume 3 Number 8 48/39/38 August 1978

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

Suppressed Operand Instructions (57)

Steve Halko (917) has been exploring ways to generate pseudos on his 57, and while he hasn't yet found a way to generate new instructions, he has discovered some interesting (and in at least one case, useful) properties of merged instructions whose operands have been suppressed. Such an instruction (SOI) is generated in LRN mode by keying SST instead of the operand. For example, in LRN mode, the sequence GTO 2 normally produces the merged code 51 2. The sequence GTO SST 2 produces 51 at one step and 02 at the next. Steve suggests writing this as GTO:: 2. Upon encountering this sequence in a running program, the machine makes an unconditional branch to the first numeral 2 appearing in the program. For example, write *00:* ***Lbl 1*** *123* ***Lbl 2*** *45678 R/S* ***Lbl 3*** *GTO:: 2* ***Lbl 4*** *GTO 2.* In RUN mode SBR 2 or SBR 4 result in 45678 displayed, as expected, but SBR 3 produces 345678, indicating that the GTO:: 2 caused a branch to the 03 at step 03 "labeled" by the 02 at step 02. As Steve points out, such behavior effectively doubles the number of available labels (for programmed unconditional branches). Users are cautioned to keep in mind that a "base" numeral used as a label executes in sequence as a numeral, and that it must be the first such numeral instruction appearing in a program to act as a GTO:: label. For example, if the 5 in GTO:: 5 appears in a program before any other 5 instruction, the GTO:: 5 effectively executes as a Nop (no operation).

SBR:: n works a bit differently. Here, the n has no effect on the operation of the SBR, which executes as a subroutine call to a sequence headed by the first zero instruction encountered in a program. If this sequence ends in rtn, and the subroutine nesting limit (2) has not been exceeded, the rtn transfers control to the step containing the n, and n itself (numeral or any other instruction) is executed as the beginning of the sequence which follows. For example, write 00: L1 1 SUM1 0 1 SUM2 rtn L2 SBR:: 1 SUM3 R/S. But before you try to run it, back up and see that the SUM3 got written as GTO3, even though you (presumably) keyed it correctly. Steve notes that the only apparent remedy is to rewrite the intended SUM3, since the keying of some SOIs affect immediately following merged instructions in such a way that the usual remedies: LRN cycling, RST, CLR, etc have no effect. Once you have the correct sequence, SBR 2 increments Reg 2 and 3, executing 1 SUM2 with the SBR:: call, and 1 SUM3 following the rtn. SBR1 from the keyboard increments Reg 1 and 2, as expected.

None of the SOIs appears to do anything useful following a conditional test, nor do a number of other SOI sequences which Steve has tried. Other 57 users are invited to investigate further, and to report significant discoveries.

Language Synthesis (58/59/PC)

It has been said that a monkey sitting at a typewriter could eventually type everything that was ever written, provided there is no limit to how long it takes, nor how much nonsense is produced. This is comparable to filling 58/59 print buffers with random numbers and printing the corresponding characters, line after line (for a very long time!), and the probability that the user would ever in his lifetime be presented with useful information would be very small indeed.

Fortunately, there is a way to significantly increase the probability of producing intelligence, and at the same time decrease the amount of garbage, and although success is enhanced by using a large, fast computer, illustrative results are possible with PPC-class machines. The idea is to take existing intelligent text (in any language), and record character and character-string occurrences in some organized manner, and use this information to weight a random selection process. The longer the baseline character strings, the greater the likelihood of producing intelligence. This approach is familiar to statisticians working with so-called Markov Processes, and is lucidly and entertainingly explored for undergraduate students by Yale Professor W R Bennett, Jr in "Scientific and Engineering Problem Solving With the Computer" (Prentice-Hall, 1976 pp 104-123). One of Prof Bennett's examples shows how an English baseline derived from Shakespeare's Hamlet (Act III) can be used to generate rather remarkable sequences of quasiintelligence formed with the alphabet, space, and apostrophe characters. Even "first-order" processing shows a significant improvement over purely "random" selection (zero order), and can be quite easily mechanized on a 59/PC, and with some data packing, on a 58/PC.

In the program which follows, Reg 1-28 contain printcode for the 28 characters, while Reg 29-56 contain corresponding counts of the number of times each character occurs in Act III of Hamlet. Processing is initiated with a call to A, with a displayed integer (n) in say the 1-1000 range serving as both an initial random number generator seed, and the specifier of how many lines are to be printed. A random number in the 1-35224 (the sum of all the character counts) range is generated, and the individual counts summed until the random number is matched or exceeded. The number of count "bins" required to accumulate the specified sum determines which character printcode to pack. This process is repeated for 20 characters per line, and n lines are printed. For example, an input of 5 produces: SHCAWID OSTUID F F NNNR S SHCAWUUU ST SYVUUIDKDYINNELRTHD' SHC'DAID FBAIDA SHCLRT OEAW SH GC EBWI in about 13 minutes. A better (more "random") RN generator might make things more interesting, but would probably lengthen the already long execution time; using ML-15 certainly would. Prof Bennett describes such first-order processing as analogous to placing before the tireless monkey a custom-built typewriter with 35224 randomly located keys: 6934 spaces, 3277 Es, 2578 Os, etc. This would, ofcourse, be about as practical to build as it would be easy to find the required monkey, and a direct computer simulation would require at least 35224 data registers. The method of summing the counts until the RN is matched or exceeded is slower, but reduces the required data registers to about 60.

```
000:  Lbl B  RCL 57 X π = INV Int STO 57 X RCL 59 = x:t 28 STO0 Lbl 1' Op20
      RCL*0 + INV x≥t
029:  1' CLR 28 INV SUM0 RCL*0 rtn Lbl A STO57 STO58 36171716 Op4 RCL58 Op06
      7 Op17
062:  Lbl 2' 4 STO61 Lbl 3' B STO60 B EE 2 SUM60 CLR B EE 4 SUM60 CLR B EE 6
087:  SUM60 CLR B EE 8 + RCL60 = INV EE Op*61 Dsz61 3' Op5 Dsz58 2' R/S
```

Prestored Data:

```
01:  13 14 15 16 17 21 22 23 24 65 25 26 27 30 31 32 33 0 34 35 36 37
23:  41 42 43 44 45 46 2043 410 584 1099 3277 629 478 1773 1736 203 34
40:  255 1238 889 1741 2578 433 6934 27 1593 1856 2557 1014 309 716
54:  21 783 14 0 0 35225
```

Each next higher order processing increases the required number of count bins by a factor of 8, and thus second order processing requires 784 bins to provide a weighting scheme reflecting the probability of the jth character following the ith. While this far exceeds the 59's available registers, tight data packing and eliminating some of the least occurring characters can make enough room. Bennett describes second order processing in terms of 28 new custom-built typewriters, where the ith such typewriter has a j key for each occurrence in the Hamlet text of character j following character i. The monkey is given one of these typewriters, say the 27th one, corresponding to the space character, and allowed to strike one of its 6934 keys. The character he types determines what typewriter he is given for the next single character he is to type, each successive typed character determining the choice of the next typewriter. Bennettt's BASIC Second Order routine converts to the following 59-oriented algorithm:

1. Prestore 441 character successor counts for the reduced character set: A,B,C,D,E,F,G,H,I,L,M,N,O,P,R,S,T,U,W,Y,space where A successor counts go in bins B1,B2,...B21; B successor counts in B22, B23...B42;... space successor counts in B421, B422,...B441. Prestore printcode for the reduced character set in consecutive bins P0,P1,... P20.
2. Initialize a pointer i to the value 421.
3. Perform: $B_i + B_{i+1} + \ldots B_{(i+k)}$ until the sum equals or exceeds
4. a random number in the 1 to $B_i + B_{i+1} + \ldots B_{i+20}$ range.
5. Pack P_k in the next print buffer position; print a 20-charac-
6. ter line; change the value of i to 21k + 1.
7. Repeat steps 3 and 4 for new lines.

The following data (from Bennett p 118, reduced to 21 characters, and scaled down by a factor of 10) show the number of times (divided by 10 and rounded) each character follows every other character (including itself). Element 001 has a value of zero, indicating that A follows A zero times; element 002 has a value of 2, indicating that B follows A 2 (actually 19) times, etc. Down near the end, element 421 heads the space successors, and shows that A follows a space 63 times, B follows a space 33 times... W follows a space 48 times, Y follows a space 25 times, and space follows itself zero times (to save paper).

21 X 21 Second Order Successor Counts

```
001:  0,2,6,7,0,2,4,0,6,14,7,42,0,2,19,16,31,2,2,11,13,3,0,0,0,14,0,0,0,
030:  1,5,0,0,3,0,3,1,0,7,0,2,1,6,0,1,0,11,0,0,9,2,2,0,0,13,0,3,1,5,2,0,
062:  1,1,3,0,0,0,11,0,1,0,7,1,0,1,10,0,2,4,0,1,0,2,66,22,0,6,12,15,2,2,
092:  1,3,16,6,26,1,4,38,22,13,0,1,3,99,5,0,0,0,7,3,0,0,2,2,0,0,12,0,4,0,
122:  2,2,0,0,23,3,0,0,0,7,0,0,6,4,2,1,1,6,0,4,2,0,2,0,0,11,34,0,0,0,63,
153:  0,0,0,26,0,0,0,19,0,1,0,4,3,0,4,21,2,0,6,4,6,5,4,0,0,10,9,35,8,1,8,
184:  29,24,0,0,0,13,10,0,0,7,16,3,0,0,11,22,1,0,16,1,0,3,2,2,0,5,25,15,
212:  1,0,0,21,0,0,0,5,1,1,1,10,2,0,2,0,5,0,12,13,4,0,8,33,15,1,16,0,3,
241:  0,0,3,22,0,0,7,12,1,0,2,41,1,2,2,5,1,19,0,0,2,5,11,27,11,3,31,7,17,
270:  49,14,1,42,5,0,0,0,7,0,0,1,3,5,0,0,6,1,6,1,1,3,0,0,5,10,0,2,11,31,
300:  0,1,0,8,1,2,2,11,1,3,9,8,4,0,5,45,4,1,2,0,23,0,0,11,7,2,1,0,12,5,0,
331:  7,23,4,2,1,79,7,0,1,0,14,0,0,88,13,2,0,0,24,0,6,5,3,5,2,3,81,1,1,3,
361:  2,4,1,4,0,2,9,2,8,0,3,20,13,11,0,0,0,19,5,0,0,0,11,0,0,16,16,0,0,3,
391:  8,0,1,1,0,0,0,0,10,1,0,0,0,3,0,0,0,1,0,0,0,14,0,0,1,0,0,0,0,48,63,
422:  33,22,23,11,26,15,45,46,24,49,24,40,21,10,48,96,7,48,25,0.
```

Only one of the original counts exceeds 3 digits: Bennett shows 1283 spaces following the letter E. So in order to keep all scaled down counts in the 1-2 digit range, I approximated the 128 for element 105 with 99. For 59 mechanization, the successor counts can be packed 6 to a register, requiring only 74 registers. This may leave enough remaining storage for packed printcode and $B_i + B_{i+1} + ...B_{i+20}$ sums, the required pointers and other temporary storage, and processing code. Devising efficient ways to retrieve the packed data via the i "pointer" will be somewhat challenging. Addressing consecutive blocks of 3 program steps in the form nn rtn would be simpler and faster, but would require reducing the number of successor counter bins by more than half.

This is a challenging, and potentially rewarding exercise, which I hope many members will attempt. I haven't yet tried to mechanize second order processing on a 59/PC, but may find that when I do, I'll want to start with a further reduced character set, in order to leave enough room for inefficient first-try processing. Readers of Bennett's book can see from scanning second order synthesized character strings how easy it is to identify the language in which the baseline text was written, even though few real English, German, Italian, or French words are actually generated. Bennett's third order examples reveal some author-identifiable real word sequences, as well as the real words themselves. As one might expect from the title, Bennett's book covers a broad range of computer applications, and I expect to address more of these vis-a-vis PPC mechanization in future 52-NOTES articles.

Membership Address Changes

5: 11A Pine St Concord, NH 03301; 45: 7211 Pine Crest Rd Catonsville MD 21228; 212: 7742 Red Lands #H2028 Playa del Rey, CA 90291; 760: 9361 NW 33rd Manor Sunrise, FL 33304; 770: Box 349 Wellington, New Zealand; 832: Box 7426 Olympia, WA 98507; 882: 10 Berryhill Rd, Greenvile, SC 29615; 998: USAF/CF Exch Off 141 Cooper St Mezz Fl Ottawa, Ont K2P 0E8 Canada.

Special case Processing (continued)

The first two challengers to the V3N7p3 program both reduced execution time by more than half, with Bill Skillman (710) leading John Mickelson (990) 7 min 41 sec to 8 min 23 sec with a revision to my V3N7p3 program. Incidently, my statement in V3N7p2 concerning a 1½ second handicap should be disregarded. I had compared Pause vs Prt execution times, but did not take into account faster 58/59 operation off the printer (V3N7p6). John's original program pause-displayed the solutions and runs off the printer in 7 min 58 sec, which my miscalculated handicap would have reduced to 7 min 56½ sec. So to compare John's program fairly with the V3N7p3 one and Bill's revision, I replaced the pause with Prt, ran it on the printer, and got the 8 min 23 sec time. But now, Bill has mechanized a new approach resulting in a 2 minute program! Both Bill and John found it useful to rearrange the given equation to: $(b^3 - 10b) + (c^3 - c) = 100a - a^3$, but Bill found more shortcuts. Both prestore the ten possible values for $(b^3 - 10b)$ and $(c^3 - c)$; John also prestores the ten $100a - a^3$ values, but Bill creates them with numeral instructions so all but the first can be examined in monotonically increasing order. John's shortcut is to bypass c-term increments when the sum of the b and c terms exceeds the a term. Bill's special ordering of the a-terms allows bypassing all the smaller a-terms when a b,c term sum exceeds the largest a-term, and all larger a-terms whenever a b,c term sum is less than an a-term, and requires only 100 b,c term sums; John's requires 1000 sums, less the number of bypassed c-terms. When an equality is found, Bill's program uses the a-term value to point to code which generates the identifier of which a-term it is for synthesis of the number to be printed. As listed below, Bill's program is sort of a rough draft, which he also used in modified form to find: $a^3+b^3+c^3+d^3=1000d + 100a + 10b + c$ solutions. As written, it gets the 6 solutions to the original problem in 2 min 1 sec; cleaned up a bit (the GTO*24... GTO... GTO replaced with SBR*24), it runs only a second faster.

TI-59/PC Program: Solutions to $a^3+b^3+c^3=100a+10b+c$ Skillman (710)

```
000:  0 GTO 109 Nop Nop Nop RCL*21 + RCL*22 = x:t 384 INV x≥t 075
025:  99 x≥t 075 171 x≥t 075 192 x≥t 075 273 x≥t 075 288 x≥t 075 336 x≥t
058:  075 337 x≥t 075 375 x≥t 075 384 x=t 103 Dsz21 007 10 STO21 1 SUM22
089:  Dsz23 007 Adv R/S
099:  1 GTO 001 STO24 GTO*24
109:  X 100 +
114:  RCL21 X 10 + RCL22 - 21 = Prt GTO 078 Lbl A 10 STO21 STO23 11 STO22 GTO
      007
171:  9 GTO 001
192:  2 GTO 001
273:  3 GTO 001
288:  8 GTO 001
336:  4 GTO 001
357:  7 GTO 001
375:  5 GTO 001
384:  6 GTO 001

                              Prestored Data:
01:  0 -9 -12 -3 24 75 156 273 432 639 0 0 6 24 60 120 210 336 504 720
```

Tips and Miscellany

An Update on TI CROMs: Following are the currently available CROMS: *1) Master Library, 2) Applied Statistics, 3) Real Estate/ Investment, 4) Surveying, 5) Marine Navigation, 6) Aviation, 7) Leisure, 8) Securities, 9) Business Decisions*, to be generally available by mid August. TI is currently putting the finishing touches to *10) Math Utilities*, expected to be available by about September or October.

I've had a glimpse at the scope, and am fairly optimistic that program quality will be better than usual. In addition to strictly math routines: Prime and random number generators, hyperbolic trig, Newton zeros, Romberg integration, Runge-Kutta DE solver, ... there will be such programming aids as an efficient Shell-sort (99 reverse-order numbers in 7½ minutes), prompting and printing aids, data packing, plotting, and if there is room: calculator status recording (V3N2p2), max/min of functions, and special-case matrix manipulations. I expect to review this new CROM in greater detail as soon as it becomes available to run.

An Update on TI's Customer Relations Telephone Service: Tom Wysmuller (743) reported what he thought was a change in TI policy: It appeared that his toll-free call to (800) 858-1802 was transferred to a cognizant person normally reachable only by means of the non-free Technical Assistance number: (806) 747-3841. It turns out that the Customer Relations office staff now includes a couple of technically oriented people, and that Tom was transferred to one of them.

Battery Pack Interchangability: James Doman (473) raises the question; TI Customer relations reports that 58/59 packs (BP-1a) may be used in 52/56 machines, but the latter's packs (BP-1) will not deliver enough current for 58/59 use.

Machine Reaction to High Humidity: Bill Skillman (710) suspects that some of his (old) 59's erratic behavior may be due to exposure to high humidity, especially since on one occasion he *"... held the open back in front of the air conditioner and Voila! it came up normal in about 3 minutes."* Bill also reports increasing instances of malfunctioning key-buttons; either no, or double results, but with the usual tactile feedback, so it's hard to tell when such malfunctions occur. Other members experiencing similar or related machine behavior are invited to share their findings.

Club Program Exchange: Dave Johnston (5) reports that the movers can't locate his household goods, including PPCs, typewriter, and pro- grams. So Exchange users are asked to be patient, and wish Dave lots of luck.

More on Dummy Operations (V2N12p3): Dick Blayney (610) notes that STO, SUM, etc does work under 58/59 program control to supply a dummy variable, provided a register address precedes the operator. For example 5 + STO 7 = puts 5 in reg 7 and displays 10. This holds for the 57 as well.

10 y^x Vs INV log: John Mickelsen (990) has been comparing the accuracy of the 2 ways to raise ten to integer values, and reports that 10 y^x is always better than INV log. John's findings appear to apply to all TI PPCs but the 57. Worst case for 10 y^x is at x=±67, and for INV log at x=±95. For this experiment, the exact solutions are easily gen- erated via 1 EE x, or multiplies of ten thereof; for non-integer x, the "correct" y^x would need to be calculated to say 14 or 15 places by an extended precision method. for the 57, results appear to be the same for either 10 y^x or INV log, and accuracy appears to decrease monotonically as the absolute value (Abs) of x increases.

Calendar Printer Status: Panos is currently leading with a 2 min 39 sec per year program; details next month.

An R/S or rtn Quirk (58/59): Rusty Wright (581) found that the manual keying of some of the complex functions: D.MS, Σ+, etc following a program halt requires keying R/S twice to get the program running again.

Volume 3 Number 9 48/39/38 September 1978

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

Roots of Polynomials

For some months now, Bill Skillman (710) has been trying various approaches to developing a general purpose root-finder for second, third, and fourth degree polynomials (quadratics, cubics, and quartics, respectively). Successive new approaches have been motivated by either Bill or my finding special cases for which earlier approaches wouldn't work. Bill's latest version (July 16th) appears to work for all of our test cases, and is listed below as a "strawman" for members to test and/or improve upon.

Root-finding is an important topic in mathematics and engineering, but one loaded with pitfalls which an understanding of some of the key mathematical concepts can help to avoid. The tutorial which follows addresses the topic of quadratic roots, and is an attempt to air some of the more important concepts vis-a-vis PPC application. It is intended to be a compromise between pedagogical rigor and grade school basics, so as to be useful to most members. I invite members with equation theory expertise to correct errors, share comments, and/or contribute sequels which extend the discussion to higher order polynomials.

The roots of any function on one or more variables are the values given the variable(s) for which evaluation of the function produces zero. ML-08 is a sort of general purpose, brute-force approach for functions in one real variable, requiring either a good initial guess bracketing each root, or lots of iterating to get some modicum of accuracy. Only real roots are found, and there is no guarantee that none is missed.

Finding the roots of polynomials is a special case, for which more efficient methods can be used, provided the user is aware of the potentially large field of pitfalls. Generally, the lower the degree of the polynomial, the easier it is to devise a closed (non-iterative), unconditional solution. An nth degree polynomial has exactly n roots, but may appear to have fewer when some are identical. All or some of the roots may be complex (grammatically "compound", since each complex number is composed of 2 parts: real and imaginary, the latter being the coefficient of $\sqrt{-1}$, called i by mathematicians, and j by electrical engineers).

Zero and first degree polynomial roots are sort of trivial special cases: Ax^0 is just the constant A, and has a "root" only if A=0; the first degree: Ax + B has the single root x=-B÷A, A≠0. (If A=0, the given polynomial reduces to degree zero).

Efficient ways of finding the roots of higher order polynomials generally involve clever rearrangement of the given terms, and the addition of new ones, to reduce the degree, step by step. The familiar quadratic formula derives from first dividing both sides of: $Ax^2 + Bx + C = 0$ by A, and rearranging to $x^2 + Bx/A = -C/A$, then adding $B^2/4A^2$ to both sides (to complete the square on the left) producing $(x+B/2A)^2 = B^2/4A^2-C/A$, which rearranges to $x=(-B\pm(B^2-4AC)^{½})/2A$ after taking the square root of both sides. The roots of a quadratic are either both real or both complex, depending on the sign of the B^2-4AC term, known as the discriminant (d). When d is negative, $x_1=-B/2A+(i(Abs(B^2-4AC))^{½}/2A$; $x_2= -B/2A-(i(Abs(B2-4AC))^{½}/2A$, or in simplified complex number notation: $x=a\pm ib$, known as a complex conjugate pair. A straightforward approach to mechanization of the quadratic formula with PPCs would be to use one type of processing when d is negative, and another when it is not. At best, this wastes time testing, and code for separate paths, compared with a single method which handles both cases, if such can be found. It turns out that the polar/rectangular functions are the key to a single method, as the V2N2p2 routines show. Here with x (Reg0 or t) set to d and y (the display) set to zero, the R-P function yields r (Reg0 or t)= Abs d and theta (display) = 0 if d is positive; -180 or 180 if d is negative (degree mode). Dividing theta by 2, taking √|d|, and converting back via P-R produces y=√|d| with x=0 if d is negative, and y=0 with x=√|d| for d not negative, providing a handy way to present either 2 real roots, or a complex conjugate pair with the same processing.

But just having an efficient routine won't always guarantee good results. When working with real numbers it is important to keep an eye out for critical data dependencies. For quadratics, when the 4AC term is small (but not zero) compared to B^2, √d can easily be indistinguishable from B, even though 4AC is measurable, resulting in the smaller root miscalculated to be zero. The best approach to detecting this problem depends on whether the user expects to catch troublesome inputs by eye, or expects the machine to flag potential trouble. Solving the problem depends on by how much B^2 and 4AC can be expected to differ in relation to machine precision, among other things, and members are invited to suggest viable approaches.

TI-59/(PC-100A) Program: Quadratic, Cubic Quartic Root-Finder

Bill Skillman (710)

User Instructions:

For roots of $Ax^2+Bx+c=0$: Key i, press i; repeat for i=A,B,C; key A'; for roots of $Ax^3+Bx^2+Cx+D=0$; Key i, press i; repeat for i=A,B,C,D;press B'; for roots of $Ax^4+Bx^3+Cx^2+Dx+E=0$; Key i, press i; repeat for i=A, B,C,D,E; press C'. Without printer connection: Press R/S following each output root; imaginary parts of complex roots are flashed. With printer connection: Real roots and real parts of complex roots are printed and tagged with R; imaginary parts are tagged with I.

Program Listing:

```
000:  GTO 620 Lbl E' SBR623 rtn Lbl D' 1 Exc1 1/x Prd2 Prd3 Prd4 rtn Lbl B'D'
025:  35 Op4 RCL2 ÷ 3 = STO8 x² +/- + RCL3 ÷ x:t 3 = STO10 RCL4 + RCL8 X (x² X
      2 - 0
059:  x:t = STO11 x² + RCL10 X x² X 4 = INV x≥t 147 √x + x:t RCL11 = ÷ 2 +/-
085:  + SBR126 x:t = SBR126 + x:t - RCL8 + ifflg2 142 E' RCL8 = ÷ 2 +/- +
112:  x:t = X 3 √x = x:t – RCL8 = GTO615 (STO13 Op10 Exc13 Abs INV yˣ 3 X
138:  RCL13) rtn 0 = STO21 rtn RCL10 +/- √x X x:t 2 = STO12 RCL11 ÷ 2 ÷ x:t yˣ
      3 =
166:  +/- Rad INV Cos ÷ 3 = STO9 SBR196 STO21 SBR187 STO22 2 X 2 X π ÷ 3 +
      RCL9
195:  = Cos X RCL12 - RCL8 = STO23 ifflg2 639 GTO E' Lbl A x:t 4 Op17 CMs x:t
220:  STO1 x:t 13 RST Lbl B STO2 x:t 14 RST Lbl C STO3 x:t 15 RST Lbl D STO4
      x:t 16 RST
250:  Lbl E STO5 x:t 17 RST Lbl A' D' 35 Op4 RCL2 ÷ 2 = +/- STO6 STO7 x² –
      RCL3 = CP x≥t
```

```
282:  296 +/- √x x:t RCL6 Ifflg4 516 GTO 615 √x SUM6 INV SUM7 RCL6 Ifflg4 491
308:  E' RCL7 GTO E' Lbl C' RCL2 CP x=t 469 D' Prd5 RCL2 X RCL4 - 4 X RCL5 =
      Exc3
337:  STO15 X +/- Exc2 STO14 4 - RCL14 x² = X RCL5 - RCL4 x² = Exc4 STO16
      Stflg2 20
367:  STO0 SBR025 INV Stflg2 CP CLR Op20 RCL*0 + RCL14 x² ÷ 4 - RCL15 = SBR581
394:  INV x≥t 376 √x STO19 +/- + RCL14 ÷ 2 INV Prd*0 = STO2 RCL14 X RCL*0 –
      RCL16 =
422:  Op10 X (RCL*0 x² - RCL5) SBR581 INV x≥t 376 √x = STO20 +/- + RCL*0 =
      STO3
451:  SBR261 RCL19 SUM2 SUM2 RCL20 SUM3 SUM3 GTO 261 RCL4 INV x=t 321 RCL3
      x=t
478:  540 RCL5 Exc3 STO2 Stflg4 GTO 260 SBR496 RCL7 CP x≥t 511 +/- √x x:t SBR
504:  616 +/- x:t GTO 616 √x E' +/- GTO E' x:t INV P/R ÷ 2 = x:t √x x:t R/R
526:  x:t STO6 SBR609 x:t 1 +/- Prd6 GTO 609 RCL5 ÷ RCL1 = x≥t 569 +/- √x √x
      x:t
553:  35 SBR620 +/- E' +/- x:t SBR616 +/- GTO E' Nop ÷ 4 = √x √x X x:t 1 =
578:  GTO 527 Fix4 EE INV EE INV Fix rtn
609:  35 Op4 RCL6 E' Stflg3 24 Op4 x:t Op6 Op8 INV Ifflg3 637 Op69 INV
635:  Stflg3 R/S CE rtn      (Write partition: 639:39; record: 479.59)
```

Calendar Printer Competition (continued)

As reported last month, Panos has pulled ahead of Lou (whose V3N7p4 program is his best so far), averaging 2 min 38.6 sec per year over the 1972-1976 5-year span with the program listed below. Panos takes advantage of Lou's "2-trick" (using 2 to both supply the integer part of HIR-formatted printcode, and a required pointer increment), and worked out a clever way to combine calculations for the number of Saturdays in a month with determining the start day of the next month: With Sun=0, Mon=1, ...Sat=6 convention (calling Pgm 20 with Reg2 set to zero does the trick), the sum of the start weekday and the number of days in the month, divided by 7 gives the number of Saturdays as the quotient, and the start day of the next month as the remainder. He also speeds up last line processing by reverse-order printcode packing, proceeding backwards from the next month start day.

My only contribution to Panos' program was to add the Lbl C entry point to provide the option of beginning with a specified month. I expect that any attempt to provide the more restrictive option of printing only one specified month, without duplicating a lot of the processing code would slow down multiple month execution unacceptably. The only labels used are B and C, and since neither is referenced by the program, they can be easily changed to suit the user's preference.

TI-59/PC Program: Calendar Printer — Panos Galidas (207)/Ed

User Instructions:

For 1 whole year: Key year, press B; for 1-9 years: key yyyy.n, press B (n=number of years); for any number of additional years (N), key N, press R/S after initial printing has halted; to start at a specified month, key year, press C, key month, press R/S, key number of years (optional), press R/S (not optional).

Program Listing:

```
000:  SUM0 RCL*0 HIR5 SUM0 RCL*0 HIR6 RCL22 HIR36 Op20 RCL*0 SUM0 INV Int
      HIR7 RCL*0 HIR8
030:  Op5 Dsz4 000 RCL2 CP x=t 121 SUM0 Op0 x:t 2 x=t 114 5 x=t 095 Op20
057:  6 x=t 087 1 x=t 091 3 x=t 075 RCL*0 EE GTO 097 RCL*0 EE GTO 105 RCL*0
083:  EE GTO 116 RCL*0 INV SUM0 EE HIR8 RCL*0 INV Int HIR7 Op30 RCL*0 INV
      SUM0 +
109:  RCL22 = HIR6 RCL*0 HIR5 Op5 CLR Adv RCL1 x:t 19 x=t 309 Op21 29 STO0
      RCL*1
138:  Int STO7 + RCL2 - (CE ÷ 7) Int STO4 X 7 = x:t RCL*1 HIR8 RCL3 Op6 R24
      HIR5 RCL25
```

```
171:  HIR6 RCL26 HIR7 RCL27 HIR8 Op5 Op0 x:t Exc2 x:t 5 x=t 026 3 x=t 018 2
      x=t
199:  008 0 x=t 002 op30 6 x=t 026 4 x=t 018 GTO 002 Lbl B  STO5 Int STO3 INV
226:  SUM5 10 Prd05 1 ST01 0 STO2 Pgm20 SBR 086 - (CE ÷ 7) Int X 7 = STI2
255:  Op23 8 STO1 RCL20 STO9 RCL3 ÷ 4 = STO4 INV Int CP INV x=t 132 Op29 RCL4
      ÷ 25
255:  = STO4 INV Int INV x=t 132 RCL4 ÷ 4 = INV Int x=t 132 Op39 GTO 132
309:  Dsz5 255 Op23 RCL3 R/S STO5 GRO 257 Lbl C STO6 STO3 R/S STO1 R/S . ST05
      0 STO2
339:  Pgm 20 SBR 086 - (CE ÷ 7) Int X 7 = STO2 RCL6 ST03 7 SUM1 GTO 260
```

Prestored Data:

```
08:  31.00000251331 29.00000211714 31.00000301335 30.00000133335
12:  31.00000301345 30.00025413117 31.00025412745 31.00000134122
16:  30.00036173337 31.00000321537 30.00000313242 31.00000161715
20:  28.00000211714 0 98.99 0 1.00003600003 1.00000037 1.0043000037
27:  1.000021000036 2.010000000002 2.010002000003 2.010003000004
31:  2.010004000005 2.010005000006 2.010006000007 2.01000700001
35:  2.010010000011 2.010011000012 2.010012000201 2.010201000202
39:  2.010202000203 2.010203000204 2.010204000205 2.010205000206
43:  2.010206000207 2.01020700021  2.010210000211 2.010211000212
47:  2.010212000301 2.010301000302 2.010302000303 2.010303000304
51:  2.010304000305 2.010305000306 2.010306000307 2.01030700031
55:  2.010310000311 2.010311000312 2.000312000401 2.000401000402
59:  2.010402
```

Roman Numeral Programs

Here is another programming topic which seems to be gathering growing interest, probably not so much because users find many practical applications, but because of the programming challenge. Dix Fulton (83) has a PPX-59 program (which I haven't yet seen) which converts "Arabic" to Roman and vice versa, and Bob Petrie (632) has a program which converts up to fairly large (1-3999999) "Arabic" integers to Roman numerals. I put the word Arabic in quotes since the numerals 0-9 are really of European origin. Bob's program lists below, and follows the modern Roman numeral conventions: I=1, V=5, X=10, L=50, C=100, D=500, M=1000, V=5000, X=10000, L=50000, C=100000, D=500000, and M=1000000, printing 2 lines in cases where the number of characters exceeds 20. There is quite a bit of "bookkeeping" involved, and Bob's mechanization is worth examining in detail to see how it works. try bettering Bob's memory requirements, or execution speed, and/or try adding a Roman to Arabic conversion. Perhaps some members will want to consider one or more earlier "Roman" conventions, such as a string of C's followed by a vertical bar and an equal number of backward C's to represent a large power of ten. For example, with a typewriter/PC character approximation: (/) =1000, ((/))=10000, (((/)))=100000, etc. The conventions for forming intermediate numbers have also evolved over the years: IX for 9 succeeded VIIII, and sometimes 19 was written IXX and sometimes XIX. So before you start writing a Roman Numeral program, decide just which applicable conventions you are going to use, and what range of numerical values to allow.

TI-59/PC Program: Roman Numerals R.G. Petrie (632)

User Instructions:

Key an integer in the 1-3999999 range, and press A. In from 15 seconds to 2 minutes see the equivalent Roman Numeral representation.

Prestored Data:

```
31:  1 100 10000 1000000 100000000 1.1 11.2 111.3 12.2 2.1 21.2 211.3
43:  2111.4 13.2 24 42 44 27 15 16 30 42.2 44.2 27.2 15.2 16.2 30.2
58:  3732320014 2422000000 4131161721 2431171600
```

Program Listing:

```
000:  Lbl A STO30 3 Op17 CMs 6 Op17 CP RCL30 x:t x≥t 250 4 EE 6 INV EE x:t
024:  x≥t 217 Prt Adv Log Int + 1 = ST09 18 ST01 24 STO2 31 ST03 30 STO7
052:  SBR230 CP x=t 142 + 34 = STO4 RCL*4 INV Int X 10 = STO5 RCL*4 Int STO6
      6
081:  STO7 SBR230 + 44 + RCL0 X 2 = STO8 RCL*8 Int X RCL*3 = SUM*2 RCL*8 INV
      Int X
111:  RCL32 X RCL*3 = SUM*1 Op23 36 x:t RCL3 INV x=t 138 31 STO3 Op21 Op22
      Dsz5
140:  083 Op20 RCL9 x:t RCL0 INV x=t 048 Op0 RCL23 Op3 RCL22 Op4 Op5 RCL29
      Op3
169:  RCL28 Op4 Op5 RCL21 Op1 RCL20 Op2 RCL19 Op3 RCL18 Op4 Op5 RCL27 Op1
      RCL26 Op2
201:  RCL25 Op3 RCL24 Op4 Op5 Adv Adv Adv Adv CLR R/S RCL58 Op0 Op1 RCL59 Op2
227:  GTO 209 RCL*7 ÷ 10 CP + x:t = Int STO*7 x:t INV Int X 10 = rtn 7 Op17
253:  Op0 RCL60 Op1 RCL61 Op2 GTO 209   (record 4 banks 479.59 partition)
```

Tips and Miscellany

An effective Op3mn on 2-Digit Registers (58/59): Maurice Swinnen (779) passes along a tip from the Jan-Apr 78 issue of **Display** (in German, due to S Seitz) suggesting a sequence of the form: *Dszmn ab* where Reg mn is to be decremented, and ab is the code for one of the Ins, Del, or BST pseudos. These pseudos are always skipped during program execution, effectively turning the Dsz mn into Op 3mn, except that zero is the lower decrement limit.

Efficient Number Base Conversions (52,57,58,59): Edward Nilges (972) notes that the usual methods, which examine digit strings digit by digit are unacceptably slow. One way to speed execution of the con- version of base b integers to base ten is to make use of some or all of the user-defined keys, each one corresponding to a number place. The sequence: ***Lbl A*** *X RCL1 + R/S* ***Lbl B*** *X RCL2 + R/S* ***Lbl C*** *X RCL 3 + R/S* ***Lbl D*** *X RCL4 + R/S* ***Lbl E*** *= R/S* will convert up to 5-place integers in base b to base ten, provided b^4,b^3,b^2 and b are stored in Reg 1,2,3,4 respectively. Incidently, in designing such a (or any other) routine for maximum speed, make it as special-purpose as you can. For example, for b=2, don't bother to store 16,8,4 and 2 since the routine will be shorter, and run faster with the in-line: ***Lbl A*** *X 16 + R/S* ***Lbl B*** *X 8 + ...;* for b=8, the cutoff is between 8^3 and 8^2 for choosing between storing and creating the multiplier. And don't design the routine to handle numbers larger or smaller than will ever be input. Fraction conversion can be added by: *...* ***Lbl E*** *+ R/S Lbl A' X 1/b + R/S Lbl B' X* $1/b^2$ *...* etc. Converting from base ten to base b is not so easy to speed up. Ideas, anyone?

Clearing Program Registers (52): Dick Blayney (610) notes that in cases where program memory needs to be cleared, but data preserved, one can write: *000: *STO00 Dsz 00*, and in RUN mode key: *97 STO00 0 rset RUN*. Execution halts at step 223 of cleared memory with a flashing zero displayed.

Programming a Variable EE Function (52): Dick wonders if anyone has been able to devise a program sequence which would effectively execute as a manual EE Mn. The obvious EE Rab unfortunately doesn't work, and the INV log and y^x functions often produce inexact results. Here is a somewhat clumsy way, using dynamic code generation: *000:* ***Lbl A*** *RCL 97 STO 96 RCL 02 SUM96 RCL 01 E rtn ... 216: stflg 0* ***Lbl E*** *EE 0 0 rtn.* Run with a call to A, with the desired mantissa in Reg 01 and decapower in Reg 02 in a n0m (for mn) format.

For example, 1.2 in Reg01 and 403 in Reg02 produce 1.2 D34. this approach could be used with the 58/59 machines, but with the additional partition/repartition hassle. Anyone have a better idea?

CROM Availability Abroad: Karl Meusch (924) raises the question, and according to TI, as of the end of August, the 9 modules mentioned in V3N8p5 are all being distributed throughout the world, but some to more places than others. If one dealer doesn't have what you want, try another, or write TI.

PC-100B: Karl also notes that a printer labeled PC-100B, interfacing with the 58/59 only is being marketed in Europe. TI confirms this, but had no further details.

Club Program Exchange: Dave reports that he is back in business again (V3N8p6), but that copying now costs 10¢ per page.

58/59 Service Manuals: Indications are that early versions have found their way into the hands of a few members. TI reports that it is now (Sept 7th) accepting orders for separate (one for each machine) manuals at $11.95 each, plus $1.50 handling and postage (Box 53 Lubbock TX 79408 Attn: Parts Order). Topics covered include: circuit diagrams for each of 7 versions (59), clock rates, a memory test program, and card read/write adjustments, according to Bob Petrie (632). However, Bob notes that "...details of internal timing, protocol, etc are expressly not discussed".

More on Printer Spacing (V3N4p6) and other Behavior: Bob has found that both the vertical and horizontal space buffers around each printed character, as well as the Op0 Op5 and Adv line spaces extend an integral number of dot positions from the 5 X 7 dot character matrix. Horizontal spacing is 2 dots to the right; vertical, 3 dots down. An Op0 Op5 line space spans 13 dots (3 from the bottom of the previous line plus ten), while an Adv line space spans 17. Bob finds each dot to measure .012" on a side, with .0173" vertical and .0157" horizontal average separation between dots. Bob notes an idiosyncrasy which may be common to most printers: feed tends to shift the paper left until it is restrained by the plastic guide. he also notes that a manually keyed Adv on the 59 advances paper continuously so long as the key is depressed. This behavior is common to the 56 and 58, but not the 52, which moves the paper just 17 dot positions, no matter how long the pap key is depressed.

Mag Card Read/Write Error Minimization (59): G. Higa (879) reports that initializing the program counter (IAR) to an address outside a bank being recorded minimizes read/write errors. Anyone else?

Blank Display (V3N7p6): Sven Nilsson (1048) reports having measured battery drain on his 58 as 127 ma with 1 displayed, but saving only 1 ma with the blank display (126 ma). But Bob Petrie (632) gets 175 ma with the displayed 1, so we seem to need more measurements to arrive at valid conclusions (comparing the same machines).

Membership Address Changes: 246: 69 palm Club Pompano Bch, FL 33062; 334: 3383 Knollwood Ct Las Vegas, NV 89121; 981: 5426 Mitchell Dr Dayton, OH 45431.

Correction (V3N7p5): Joel Pitcairn (514) has kindly pointed out that the minus sign (near the end) in the integer extraction routine should be removed.

Volume 3 Number 10 48/39/38 October 1978

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

The Business Decisions (9) CROM

Bob Anderson (506) got an early look at this new CROM, and reports it to be a "... super collection of programs." Bob is a business management/administration professional, not given to heaping undeserved praise in TI, so it appears that TI chose some useful business applications to address. At last report, Bob had not yet delved very deeply into program operation or structure, so quality has yet to be determined. Bob did report that a suggested (listed, but not in CROM) number sorting program (p77-78 in the manual) doesn't work. It is terribly slow, destroys the input data, but probably does what was intended: assign consecutive keys (V3N2p3) to input records, and output the keys in an ordering determined by ascending magnitude of corresponding records. It's unfortunate that the documentation doesn't specify what results to expect, and this exercise brings up an important question to ask yourself when writing/using sort routines: What is it you wish/expect to be output: keys or records, or both?

Another problem shows up in this CROM, which needs to be recognized when 2 or more CROMs are needed to solve a single problem: Since you have to turn the power off to change modules, all intermediate results are lost, unless the data are recorded. BD-3 requires ML-19 with the repeated entry of 3 inputs, as well as the BD-3 output. In this case, the user would probably find it easier to put BD-3 in user memory (it's only 60 steps) and not have to change modules, especially if more than one run is anticipated. For extended use, it would be worthwhile to add a post-processor to BD-3 code, which would initialize the required registers for ML-19, and then call it.

It is too bad that TI still doesn't program the CROM routines to sense machine configuration, and vary I/O processing accordingly. For example, BD-5 would run considerably faster if the pause-displayed outputs were automatically by-passed with printer connection, and there's no reason why the user should have to set partitioning, since it's easy enough for the program to sense whether it is running in a 58 or 59 *(... 7 Op17 Op19 ifflg7...)* and it is given the max number of rows as input.

Members with appropriate expertise are invited to review any of the CROMS from programming quality and/or technical integrity standpoints. Incidently, recent full page ads (**Scientific American** and **Electronics**) identify Math/Utilities as #10 and Electrical Engineering as #11, both to be "Coming this fall...".

Bridge Deal (V3N2p5)

Bill Skillman (710) has risen to Lou's challenge. and with the program listed below cuts execution time by more than half and saves 170 steps. The key to Bill's approach is that the cards are always put in the correct order in each hand, as they are dealt, eliminating the need for sorting. Here's the algorithm: Form printcode for each of 52 cards symbolized: SA, KS, ...S2, HA, HK, ...C2, one at a time, always in the same order. Following the generation of a card, randomly select one of 4 13-place fields and put the card on the next available position. When a field has been filled, bypass it during subsequent random selections. Continue until all fields have been filled. Print the contents of the 4 fields: 13 lines, the ith line containing the ith element of each field. fast as it is, there is a possible flaw in this approach, the seriousness of which is left to the user to assess: the more the random selection becomes uneven (one to three fields are filled well before the other(s)) the greater the probability that the remaining field(s) will be filled with consecutive low-rank clubs. Users should try an assortment of RN seeds and many successive deals to see how influential this anomaly is.

In mechanizing this algorithm, Bill does some tricky pointer maneuvering, which users should find helpful in other applications. See how printcode is synthesized via Reg 6 and 86, and try following the double indirect addressing accomplished with Reg 7 and 8. Initializing the last element in each of the 4 "hand" fields (steps 082-090) with a one and testing via the Exc* function (steps 149-153) is a short, fast way to sense when a field has just been filled; steps 159-182 effectively remove filled fields from the random selection process. My only contribution to Bill's program was to add the multiple-hand capability.

TI-59/PC Program: Bridge Deal Bill Skillman (710)/Ed

User Instructions:

```
Record banks 1 and 2 with turn-on partition.  Key seed (0-199017), press A;
key number of hands to be printed (defaults to one), press R/S.  Output is
printed at the rate of about 3.1 minutes per hand.  The ML CROM is required,
and tested for.
```

Program Listing:

```
000:  Lbl D 9 ST00 22 ST01 35 ST02 48 STO3 rtn Lbl A ST09 1 x:t Pgm1 SBR
      Write INV
029:  x=t X 9 Op17 CLR R/S STO87 RCL85 Op4 RCL9 Op6 Adv Op0 RCL83 Op2 RCL84
      Op3
057:  Op5 Adv D 4 STO5 RCL79 Op1 RCL80 Op2 RCL81 Op3 RCL82 Op4 Op5 1 STO22
      STO35 STO48
089:  STO61 x:t RCL75 E' RCL76 E' RCL77 E' RCL78 E' B' Adv Dsz87 060 rtn
      Lbl E' STO86
115:  13 ST04 62 STO6 Pgm15 SBR D.MS X RCL5 = STO8 1 SUM*8 x:t RCL*8 STO7
      RCL86 +
144:  RCL*6 Op26 = Exc*7 x=t 159 Dsz4 123 rtn Op35 3 - Exc8 Int = CP x=t
169:  154 STO7 RCL3 Op38 Exc*8 Dsz7 175 GTO 154 Lbl B' fix2 D 13 STO4 Op20
      Op21
199:  Op22 Op23 RCL*0 Op1 RCL*1 Op2 RCL*2 Op3 RCL*3 Op4 Op5 Dsz4 195 INV fix
      rtn
```

Prestored Data:

```
62:  13 26 34 25 201 12 9 8 7 6 5 4 3 360000 230000 160000 150000
79:  3132353723 17133637 3632413723 43173637 143524 1622170000 36171716
```

Print Buffer Filling with an EE or Eng Display (58/59/PC)

Bob Petrie (632) notes that filling print buffers via Op1-4 "... with an EE or Eng format yields some interesting results," and suggests that the inherent Int function is not performed. What appears to happen is that as many significant figures (up to 8) as are

in the effective integer part of the number represented in EE or Eng format are transferred right-justified to the print buffer mantissa field, along with the exponent as displayed. The sign digit assumes values of 0, 2, 4 or 6 depending on the length of the transferred mantissa string, and follows different rules for EE and Eng formats. For example, 1.3141516 D07 is the integer 13141516, and in the EE format when subject to an Op 1-4 becomes print-buffer formatted as -0000013141516 D-07 (or as an octet of step-code: 76,60,51,41,31,01,00,00 if it could be transferred "intact" to a program register). Displaying the buffer contents causes a left shift of 5, producing -1.3141516 D-12 .

Last Digit Viewers

Short routines to extract the hidden guard digits in a 52 have been around for some time now (V1N4p6 and V3N1p3), but comparable ones for the other TI PPCs haven't yet surfaced. The following discussion applies to one or more of the TI PPCs, but not the 52. If a fractured display approach can't be found, then the mantissa must be maneuvered by some other means, and all which have surfaced so far require use of the EE function, limitations about which are discussed later. As in most program-design situations, it's a good idea to decide exactly what results are defined and how the program is to be invoked, before charging off writing code. In this case it is important to decide first whether the program is to be called by another program, run directly by the user, or even just keyed manually. Steps and execution time can be saved if the user examines a displayed number, and initiates mantissa partitioning according to some simple rules. Also important in this case is the type of stack arithmetic performed. Early 56 machines truncate the 13th digit of the first of the 2 operands used by a dyadic function; later 56s truncate the 13th digit of both operands (V2N2p6). So for either model, register arithmetic is required to preserve 13 places, and one can manually extract the ten least significant digits as described in V1N7p2. But program processing would need to perform the *"... subtract it from the first 3 digits (including the indicated decapower)..."* part, and this is more cumbersome to get the machine to automate than to do by eye.

Besides concerning themselves with decapower magnitude and sign, the TI PPCs must take into account display rounding by the EE function, using it in reducing the display to remove MSDs from the full mantissa. For example, on a 58/59 at turn-on, key π - EE = and see that the resulting -4.1 D-10 is the difference between π(13) and π(10) (π(n) means π rounded to n places). Now (with the display still in sci format key π - EE = again, and see that this time π(13) - π(8) = -4.641 D-8 was performed. For both π(10) and π(8), rounding was "up" one place producing larger numbers than π(13), and the differences produced represent the so-called ten's complement of the desired LSDs of π. Now try all this with e (2.718...) instead of with π, and see that since the rounding is down (truncation) in both cases, resulting differences are equal to the desired LSDs. From these 2 examples it might appear that we have a simple method for extracting the LSDs: Perform - EE = on the display, and if the result is negative, convert the ten's complement result to its inverse (add 10^{n+1} to an n-place ten's complement number). It turns out that this will work for most reals, including those with large decapowers, but not all. The exceptions appear to be numbers whose 11 MSDs are all 9s, but whose decapowers are less than 99, and any number which when its display rounded configuration is subtracted from it produces underflow (i.e. ± π D-99,98,...92; ± e D-99,98,...92; etc).

But there doesn't seem to be a practical way to mechanize this in a program-callable subroutine, such that results are always in a predictable format. The following 58/59 35-step routine (which makes use of Dix Fulton's integer function found elsewhere in this issue) uses a different approach, and appears to work as a program-callable subroutine for all reals, returning the 9 LSDs of the mantissa as a decimal fraction, flashing zero for a zero input; ***Lbl A*** *abs STO 11 log + (INV Int - abs) Op10 = Int INV log EE INV Prd11 1 EE 3 Prd11 RCL 11 INV Int INV EE rtn.*

If you only want the whole machine number printed, and don't need to use it internally, the simplest (but not neatest) way is to list the octet of 8 steps containing the number as data. The following 58/59/PC routine is subroutine callable, but requires the user to interpret the listed steps (224-231) as data (V1N1p4,5), and prints 16 lines per real;

```
Lbl  A x:t 10 Op17 9 STO 92 9.2 STO 90 x:t STO 91 9 Op17 GTO 223.
```

The non-printer approach can be mechanized for the 57 as:

```
  Lbl 1 abs STO 1 log Ct x≥t GTO 2 + INV Int x=t GTO 2 1 +/- Lbl 2 = Int INV
log EE INV Prd1 1 EE 3 Prd 1 RCL 1 INV Int INV EE rtn
```

which returns with the 7 LSDs of the 11-digit full mantissa. Overflow reals cannot be processed, since the 57 will not execute during an error condition.

A 56 version may be written:

```
00:  abs STO 1 log CP x≥t 16 + INV Int x=t 16 1 +/- = Int x:t 10 y^x x:t = EE
INV Prd1 1 EE 3 Prd1 RCL 1 INV Int INV EE rtn
```

which returns with the 9 LSDs of the 13-digit full mantissa of non-overflow reals.

This article was motivated by inputs from Panos Galidas (207), Bob Cruse (889) and John Mickelsen (990). All members are invited to comment, and to suggest better approaches to non-52 last-digit extraction.

Some 58/59 Firmware Revealed

Steffen Seitz (1030) has found that an odd CROM calling sequence (V3N3p2) can reveal the code executed by the Σ+, x, P/R, D.MS, their inverses, and the Op 11-15 functions. With a 58 or 59 at turn-on (ML installed), key: Op9 Pgm24 R/S R/S R/S 99 Op17 GTO 0 R/S D.MS, then LRN and SST through step 487, or key List if the PC100 is connected. Steffen has identified steps 000-044 with Op12, 047-057 as Op15, 058-066 as Op14, 067-081 as x, 084-106 as Op11, 107-148 as INV x, 149-191 as Op13, 192-249 as Σ±, 250-283 as INV P/R, 284-302 as P/R, 303-340 as D.MS, and 341-379 as INV D.MS. Steps 380-487 make no sense as program instructions, and all but one octet are non-transferable (V1N2p2) if interpreted as data. Continued listing or SSTing past step 487 begins at step 039 with an abs instruction, then 040,041,... 487 are as before.

Steffen suggests that the HIR 20 instructions at steps 045-046 and 081-082 serve as rtns. If so, HIR 20 behaves differently when executed in the firmware than in user memory, where it appears to do nothing but harden a soft display. Some variations on Steffen's sequence seem to work: ML Pgms 2, 3, 4, 8, 11, 12, 13, 18 and 25 may be substituted for Pgm 24, and 9 Op17 for 99 Op17; no repartitioning is required with a 58. No other CROM module appears to work, so it seems that the linkage mechanism to get to this firmware is peculiar to the Master Library. Although the code looks like it is in user memory, it probably isn't: it can't be executed or edited, and a BST causes an infinite hangup.

Also, the 488 steps exceed the current partition, and the circular stepping (039, 040, ...487, 039, 040, ... etc) is inconsistent with user memory behavior.

Other members are invited to build upon Steffen's discovery, and to try to gain access to other sections of firmware and/or internal registers (IAR, subroutine return address stack, arithmetic stack pointer, etc).

Repeating Decimals

As a follow-on to George Hartwig's all-digits generator (V3N6p5), Jared Weinberger (221) notes that 9800 ÷ 9801 = .99 98 97 ...0. These are rounded single cycles of repeating decimals, whose unrounded cycle patterns are 987645320, and 99 98 97 ... 03 02 00, respectively. These 2 examples suggest a more general relationship: Starting with the integers 80 and 81, and calling these K_0 and K_0 +1, form K_n by preceding K0 with n nines and following it with n zeros, n=1,2,... . Then the fraction $K_n \div (K_n+1)$ is a repeating decimal having a cycle length of $(n+1)(9X10^n+9X10^{n-1}+ ...9)$ digits, having the form: 9...9 9...8 ... 0...2 0...0, each p...q group being n+1 digits long. For example, for n=1 (Jared's example) K ÷(K +1)=.99 98 ... 02 00, 99 98 etc with a cycle length of 198 digits. For n=2, we have 998000÷998001=.999 998 ... 002 000, 999 998 etc, which has a cycle length of 2997 digits. Note that in each case 0...1 is skipped. As n increases, the cycle length grows exponentially, so it is impractical to check this relationship for very large n, even with a large fast computer. Anyone have a mathematical proof?

There are, of course, other interesting (and many more not so interesting) repeating decimals (fractions whose numerators and denominators are integers, but for which when the indicated division is performed, the remainder cannot be expressed exactly as a decimal fraction). Members may find the following 58/59/PC infinite division program helpful in trying out various repeating decimal generators. To save it, record banks 1 and 4; to use it, key the dividend, press A; key the divisor, press R/S. The input problem is print-confirmed, and the quotient printed: the integer part first, followed by 20-digit lines of decimal fraction until stopped with R/S.

```
000:  Lbl A STO12 72 Op4 RCL12 Op6 ÷ R/S STO11 64 Op4 RCL11 Op6 = Int STO10
      Prt 1
030:  STO13 4 STO15 1 EE 8 STO16 INV EE 0 STO14 5 STO17 RCL10 X RCL11 = INV
      SUM12
057:  10 Prd12 RCL12 ÷ RCL11 = Int STO10 RCL*10 X RCL16 = SUM14 .01 Prd16
      Dsz17
085:  048 RCL14 Op*13 1 SUM13 Dsz15 035 Op5 GTO 029

Prestore:  00:  1 2 3 4 5 6 7 10 11 12.
```

Mechanizing the inverse: Given a repeating decimal, find the equivalent numerator and denominator integers looks like a considerably tougher problem to solve if other than a brute-force trial and error approach is taken. Members are invited to address this challenge.

Tips and Miscellany

The Math Integer Function (V2N12p1, V3N6p6, V3N7p5): Dix Fulton (83) beats the others with: *... + (INV Int - abs) Op10 = Int ...* which is both shortest and fastest, and appears to handle al cases.

TI's Sourcebook for Programmable Calculators: Kirk Gregg (748) brought this new TI publication to my attention. 58 or 59 owners who bought/but their machines between 15 August 1987 and 31 October 1978 are eligible for a free copy. Others may buy the book at $12.95 each. This appears to be a 58/59 version of the Programming workbook written for the 52 (V1N5p2), but with more applications examples. It may be useful for users of other machines. I will report further after I've seen this book.

Flag Status Routines (52,58,59): Several members have suggested multi-step routines to display the status of flags. Assuming that the purpose of these routines is to help debug other programs, I find it difficult to find an occasion for which the considerably shorter manually keyed: ifflg n 888 (V1N4p6), or ifflg n N (V3N3p6) wouldn't be better. Comments?

The Newest 59s: Dave Leising (890) reports that the newest 59s bear a code of the form: ATA nnnn (V3N4p1), and *"... contain a differend mag head which will not allow reading cards ..."* made on older machines. TI acknowledges that some calculators are assembled in Abilene (Texas), which probably accounts for the first A in the ID code. Mag card compatibility may be unit-dependent: I find that some cards recorder on older LTA machines will read (sometimes) on an ATA machine made the 23rd week of 1978. Hardware visible in the battery well appears to be the same as for LTAs of late 1977 vintage, except that 4 resistors are larger in the newer machine, and the PC board, while having the same layout, is more like the first machines (July 1977) in the appearance of the surface material. No differences in functional behavior have come to my attention yet.

Membership Address Changes: 207: 150 Monroe St #302 Rockville, MD 20850; 398: 35 Oakfields, Broad St, Guildford, Surrey GU3 3AS England; 616: 761 Via Flaminia Vecchia 00191 Roma Italy; 689: bei Hartmann Schnewittchenweg 5 D7500 Karlsruhe Germany; 998: 9 Centre Park Dr Ottawa, Ont K1B 3C2 Canada.

Machine Power Consumption (Blank Display V3N9p6**):** Sven notes that power (milliwatt) drain is a better way to compare machine states than current (milliamp) drain, since for any fixed state, the machine draws more current as the battery becomes more discharged (output voltage drops). For a displayed 1, Sven measures 515-540 mw for one pack, 530-560 mW for another, and an average 1.45% power consumption difference between blank and 1 displays.

Writing HIR Sequences (58,59): Tony Tschanz (1034) suggests sequences of the form: ...RCL 82 BST BST i SST j ... where i is the instruction preceding the HIR and J is the HIR operand. j is most easily formed by finding a function key with the required op code. For example, to write: ... + *HIR12* ..., key *RCL 82 BST BST + SST B*. This method saves the Del step required by the usual method, but is no better if i is an op code which by itself would be a pseudo. For example, ... *STO 31 HIR 3* ... written *STO31 RCL82 BST BST Del SST 3* has no more keystrokes than *SST RCL 82 BST BST BST STO 31 SST 3*.

Flashing Display Variations (V2N12p4): Tony finds that the 59 (58 too) does have an error condition display format analogous to the 52's soft flash: The sequence: CLR 1/x 0 , alternates the C symbol with the displayed zero. It appears that the soft flashing display prevails through number buildup in the display, but turns hard when the display itself is hard, unless either minus sign is present. For example, CLS 1/x 24 initiates a soft flash, which turns hard following the keying of STO. But CLR 1/x 25 +/- maintains the soft flash following STO.

Mag Card Cues (52,59): In cases where successive mag cards are to be read, but where their selection is determined by the outcome of on-going processing, Bob Petrie (632) suggests having the last processing of each card provide a cue identifying the next card to be read. The first processing of the next card compares the cue with its identity, and alerts the user to mis-matches.

Volume 3 Number 11 48/39/38 November 1978

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

More on the Revealed Firmware (V3N10p4)

Dix Fulton (83) and Gordon Wilk (1089) have been delving into Steffen's discovery, producing several new findings.

Gordon has narrowed a valid calling sequence to: Pgm 12 SBR 333 R/S D.MS, explaining that *"... The SBR address may be any address outside of the current user partition and beyond the end of the called program. Other (non-Op) functions may be substituted for D.MS if their address in firmware is greater than the last step in the current partition."* Pgm 12 is only 155 steps, and D.MS starts at step 303 in firmware (ROM), so a partition of 239 or less will work for this calling sequence. A list or LRN SST ... begins at the ROM step corresponding to the current user memory (RAM) step prevailing at the call. For example, at 59 turn-on key 9 Op 17 GTO 111 PGM 12 SBR 155 R/S P/R LRN and find ROM step 111 displayed. Gordon goes on to explain that following the R/S in the calling sequence *"... the calculator seems to execute something in the program as well as Stflg 9 EE SBR* since when it stops it is expecting a register address where it will find the program address or label to search. If the key at this point in one of the programs in (revealed) ROM it will set the IAR to this step – in RAM, if the current partition goes that far, or in ROM if the current partition is too short. If any other (non-numeral) key is pressed there is a label search ending at the end of the partition."*
I tend to agree with Gordon's explanation, except for the last statement. It appears that several seconds of some processing (which may or may not be label searching) goes on following the keying of a non-revealed ROM function, followed by execution of RAM beginning at the step current at the call. Gordon continues with: *"Since user RAM, firmware ROM and library programs all seem to begin at step 000, the IAR must actually be wider than the 3 digits of the displayed program counter. Pgm nn and the keys accessing firmware must set a value in a base register which is added to the displayed program counter to arrive at the real machine address for the IAR. Further, there must be some way of translating the keycode or program number into, respectively, starting address and base. Perhaps this is by searching registers to find an indirect address. Since steps 384-487 in firmware are 13 registers and there are 13 programs in this firmware, these are addresses and labels - the code is certainly not obvious.".* The code is, indeed, not obvious. Anyone care to attempt a translation?

Dix notes that nothing in either the D.MS or INV D.MS code explains the rounding which occurs on a fix n display. Indeed, if the revealed D.MS or INV D.MS code is keyed into RAM and executed, there is no rounding. Upon inspection of the code containing the 2 HIR 20 instructions, Dix found that as executed in ROM, HIR 20 appears to behave in 3 ways: rtn, Nop, and GTO*. He arrived at this conclusion by suggesting *"... that Op 12, Op 14, and Op 15 all cause execution to begin at location 000, with a HIR 20 jump from location 045/046 to location 058 for Op 14, and HIR 20 behaving as a Nop for Op 15. Similar behavior must occur for the HIR 20 at 082/083, enabling Op 11 and x to share the same code."* However, it would seem that for HIR 20 to function only as a GTO* instruction would be consistent with all the observed behavior. Assuming that each of the 3 Op functions initializes the HIR 20 GTO* pointer before execution, Op 12 would specify 57, causing a transfer at step 045/046 to the rtn at step 57, Op 14 would specify 58, and Op 15 47. Similarly, x could specify 57 (or any other step containing a rtn instruction), and Op 11 84.

I find that at least one of the revealed-ROM functions: INV D.MS sometimes performs arithmetic at a higher precision when executed in RAM. For example, ROM D.MS INV D.MS on 2.5614 produces 2.561399999920 while when executed in RAM produces 2.561399999999. The intermediate 2.937222222222 following D.MS on 2.5614 is the same either way. This difference in precision suggests either that the arithmetic unit interprets ROM code differently than the same RAM code, or that ROM arithmetic is performed in a separate arithmetic unit. The same HIR stack is used either way, but the residuals in HIRs 1, 2, and 8 differ when ROM and RAM INV D.MS results differ.

The D.MS code does confirm the algorithm suggested in V3N7p5 to determine the H8 residual, and the INV D.MS code can be used to answer the question of the H2 and H8 residuals.

Fast Number Base Conversion Routines (V3N9p5)

Bill Skillman (710) has addressed the base ten to base 8 problem for integers, and devised one approach for the 59/PC using prestored values for base 8 characters to speed up processing. Although such table-lookup schemes can be helpful in speeding execution in some applications (see some of the faster calendar printers in recent 52-NOTES), they may not be very effective for fast number base conversions. It turns out that the straight forward iterative reduction by division with repeated separation of quotient and remainder, of a base ten number (integer) by base b is considerably shorter and faster than Bill's method, especially if the output base b number is processed one numeral at a time.

In this discussion I use the word numeral to designate a "place" in the positional representation of numbers, avoiding the use of "digit" which implies limitation to one of the 0-9 symbols. For example, the largest base sixteen numeral is usually denoted by F, but would appear as 15 in a base ten display. MSN means most significant numeral; LSN least significant numeral.

Here is a short fast (but slower than base b to base ten (V3N9p5) base ten to base 8 integer converter written for the 58/59, but easily modified for the 56 or 57: ***Lbl A*** *STO 0 CP CLR RCL 0* ***Lbl 1'*** *÷ 8 - Int STO 0 = X 8 = Pause RCL 0 INV x=t 1' R/S.*

Key the base ten integer, press A and see the corresponding base 8 numerals pause-displayed from LSN to MSN.

Conversion of positive reals is just as straight forward, provided the integer and fractional parts are input separately. For example: Lbl B X = - Int Pause = GTO B may be added to routine a, and run with an R/S or B after the base ten fraction has been keyed. The corresponding base 8 fraction is paus-displayed, one numeral at a time, starting with the MSN, and continuing indefinitely to the right of the radix point until stopped with R/S. For example, to convert the base ten real: 123456789.123456789 to base 8, key the integer part, press A, and see pause-displayed: 5,2,4,6,4,7,6,2,7 representing the base 8 integer: 726746425. Then key the fractional part, press R/S, and see: 0,7,7,1,5,3,3,5,1,5,6,3,4,5,4,3,2, ... representing the base 8 fraction: .07715335356345432 ... .

Generalizing these routines to handle other number bases in addition to octal is easy enough, and does not slow execution speed perceptibly. For example:

```
Lbl E STO 1 R/S LD CP STO0 LBL 1' RCL0 ÷ RCL1 - Int STO 0 = X RCL 1 = Pause
RCL 0 INV x=t 1' R/S Lbl 2' X RCL 1 - Int Pause = GTO 2' LBL B 1 STO 0 CLR
R/S LBL3' X RCL 0 ÷ RCL 1 Prd00 0 R/S GTO 3' LBL B' RCL 1 STO 0 CLR R/S LBL
4' ÷ RCL 0 + RCL 1 Prd00 0 R/S GTO 4'
```

handles conversions both ways, and is run by first keying the base (positive integer), and pressing E. (Do this once per base specification). For base ten to base b, key the integer part, press D, and see the base b integer pause-displayed numeral by numeral, beginning with the LSN. Key the fractional part, press R/S, and see the base b fraction pause-displayed numeral by numeral beginning with the MSN; stop with R/S. For base b to base ten, initialize by pressing B, then input each numeral of the integer part (2 or more digits for b greater than ten), followed by R/S, starting with the LSN; end with =, and see the entire (up to ten-place) base ten equiIvalent integer displayed. Key each numeral of the fractional part, followed by R/S, starting with the MSN; end with =, and see the entire base ten equivalent fraction.

When designing such routines for printer connection, the user needs to trade the relative advantages of speed versus paper economy: The fastest will use the most paper (and not be so easy to read) as slower routines which pack the input or output base b numerals before printing them. 58/59/PC routines designed for base ten to base sixteen conversion could output 4 groups of 4-numeral hex characters, closely resembling the memory dump format of many computers. Expediting input for the inverse operation: Hex to ten poses somewhat of a problem, although use could be made of the A-E and A' keys for the A-F numerals.

Members are invited to share their best number base conversion programs, especially much-used special-purpose ones which work well in real-world applications, and demonstrate useful programming techniques.

Friendly Competition

In June 1977 when the new 58/59 machines had been announced, I stated that HP would have to catch up before the 58/59 could fairly enter the Friendly Competition arena (V2N6p4). Now I find an area: execution speed, where the 58/59 may be hard put to match the HP-67, since some arithmetic is faster on the HP-67. This came to my attention when I translated an optimized Factor Finder program written for the 67 (**PPC Journal** V5N2p22) into 59ese, and found that it took about twice al long to run on a 59!

It turns out that for large numbers, finding the prime factors of one integer is harder and more time consuming than finding the greatest common divisor of 2 integers, or testing an integer for primality. Quite a few shortcuts have been devised, with perhaps the most comprehensive analysis of factoring techniques under one cover written by Donald Knuth in Vol II of his **"The Art of Computer Programming"**. But most methods in the literature work best on large fast machines, processing very large numbers. A method which lends itself well to PPC mechanization first tests an input integer for division by 2, 3, and 5, dividing out by these factors and their multiplicities if/when there is no remainder. Successive divisions are performed using increasingly larger divisors, skipping those which are multiples of those already tried, until the divisor exceeds the square root of the remaining integer. For HP-67 implementation, the code exercised the most often takes the form: *RCL 2 RCL 3 ÷ Frac x=0 GSB 0 RC S+3*, which takes about .38 seconds to execute. Corresponding 59 code looks like: *RCL 2 ÷ RCL 3 = INV Int x=t 1' 4 SUM 3*, which takes about .53 seconds to execute.

So the challenge is to beat J L Horn's HP-67 program, which he claims proves 9999999967 prime in 2 hours 55 minutes (worst case for a maximum of ten digits). A smaller test case supplied by Lynn Yarbrough (1081) with a 58 Factor Finder "assembled" by a program he wrote in SNOBOL, is to factor 987654321 into 3,3,17,17,379721. Lynn's program takes about 5 minutes, Horn's about 1 minute, and the following 58/59/PC program adapted from one Mike Louder (53) wrote for the 52 (**65-Notes** V3N4p9) takes about 2 minutes. This turned out to be slightly faster than my 59 version of Horn's 67 program.

TI-58/59/PC Program: Speedy Factor Finder Mike Louder(53)/Ed

User Instructions:

Key a positive integer, press E. See printed input confirmation, and the prime factors.

Program Listing:

```
000:  R/S Lbl C SUM2 RCL1 ÷ RCL2 = INV Int x=t 2' rtn Lbl 3' CP 6 C 4 C 2 C 4
      C
027:  2 C 4 C 6 C 2 C RCL 2 x:t RCL3 x≥t 3' RCL 1 Prt Adv Adv Adv CLR RST
051:  Lbl 2' RCL1 ÷ RCL2 Prt = STO1 √x STO3 0 GTO C Lbl E CP STO1 Prt Adv √x
      STO3 1
079:  STO2 1 C 1 C 2 C 1 STO2 GTO 3'
```

Repeating Decimals (V3N10p5)

George Hartwig (638) and Samuel Allen (1032) note that a repeating decimal: .a1 a2 ... an a1 a2 ... may be written in rational fraction form as: (a1 a2 ... an)/(99 ... 9), where the denominator is composed of n 9s. Samuel notes further that "If a decimal fraction repeats only after a certain number of places, it may be broken up into the repeat- ing part plus the non-repeating part, the two expressed as rational fractions and added; this .abcdefgefg... could be expressed as the sum of abcd/10000 and efg/9990000" which generalizes to: .a1 a2 ...an b1 b2..bm b1..=((a1 a2 ...an)/10^n) + ((b1 b2 ...bm)/ (99...900...0), where the denominator of the second term is composed of m nines and n zeros. These 2 terms combine to: ((99...9)(a1 a2 ...an) + (b1 b2 ...bm))/ (99...90...0).

When both the numerator and denominator are short enough to fit in a PPC register, reduction of the rational fraction to lowest terms is quick and easy via Euclid's Algorithm (which finds the greatest common divisor (GCD) of 2 integers). For example, .865259740259740... may be expressed as 865258875/999999000, which reduces via the GCD of 1623375 to 533/616: Key the denominator, press A; key the numerator, press R/S, and see the GCD in about 8 seconds, using the following 58/59 mechanization of Euclid's Algorithm:

```
Lbl A CP STO 1 ÷ R/S STO 2 Lbl 1' = INV Int x=2 2' X RCL 2 STO 1 = EE INV EE
STO 2 RCL 1 ÷ RCL 2 GTO 1' Lbl 2' RCL 2 R/S.
```

But if a numerator or denominator is too big to fit in a PPC register, reduction to lowest terms would require extended precision arithmetic, which in a 59 might be made to handle up to 500-digit numbers, but would be slow. So for long-cycle cases, other methods might be more attractive. One way is to take a quasi-brute-force approach: Start with the minimum which the denominator can be (one larger than the number of digits in the repetition cycle), and a numerator the nearest integer to half the denominator. Divide, and compare with a full-register representation (truncated 13 MSDs for the 58/59) of the repeating decimal. If the result is too large, cut the numerator in half, if too small, increase it by half. Compare, and repeat the process, taking the mean of bracketing numerators until either a perfect comparison occurs (stop), or the brackets meet, in which case add one to the denominator and start again with a new half-denominator numerator. If (as is likely) the input repeating decimal cycle is longer than a full-register length, test the results with an infinite-division routine (V3N10p5). If the results are incorrect, add one to the last denominator and start again. An upper limit for this approach using a 52/56/58/59 would appear to be a repeating decimal whose cycle length is around 10^{12} digits, and whose rational fraction lowest-term denominator is the smallest it can be.

This method is quick when the denominator of the rational fraction is at or close to the minimum, but very slow for cases like 533/616, where the minimum denominator appears to be 7, although the 3-digit preface to the first repetition cycle argues somewhat intuitively for a larger minimum denominator. But how much larger?

On the other hand, this method is very quick to determine that a 1570-digit repeating decimal cycle which begins with 8650541056651... and ends with ...9847231063017 is equivalent to the rational fraction 1359/1572: Prestore the 13 MSDs in Reg 1, key the minimum denominator (1571), press A, and get the printed (numerator in T, denominator displayed if no printer) rational fraction in 17 seconds with the following routine:

```
000:  Lbl A STO2 Lbl 5' ST04 0 STO3 Lbl 1' RCL1 x:t RCL 3 ÷ RCL 4 = ÷ 2 = Int
      STO5 ÷ R2=x≥t
033:  2' RCL5 - Exc3 = CP x=t 3' GTO 1' Lbl 2' x=t 4' RCL3 x:t RCL5 x=t 3'
      STO4
058:  GTO 1' Lbl 3' Op22 RCL2 GTO 5' Lbl 4' RCL5 Prt x:t RCL2 Prt Adv Adv Adv
      R/S
```

This contrasts with more than 17 minutes required to find that a 78-digit cycle beginning with 8662420382165... and ending with ... 2611464968152 is equivalent to 136/157. But my guess is that using the Euclid Algorithm approach with extended precision arithmetic wouldn't be any faster, especially considering the time required to input 156 digits (correctly!). Members are invited to explore repeating decimals further, and to share their discoveries, insights, etc.

Tips and Miscellany

Calculator RF Interference: Brian Agron (954) notes that Federal Aviation Regulation 91.19 prohibits the use of any electronic device on most flights unless the device does not interfere with navigation or communications. This would appear to put the burden of proof on the user. Brian also passes along a clip from FLYING (Sept 1978) which indicates that the Canadian government has determined that 5 different calculators, each operated within 3 feet of an ADF loop antenna, seriously interfered with reception in the 200-450 KHz band. PPC operation in the cabin of an airliner is probably too far from sensitive flight equipment to do any harm, but operation in the cockpit of any aircraft might cause trouble. So when in doubt, check out an intended configuration of calculator and avionics before flying.

Used SR-52s and SR-56s: S.G. Allen (1032) would like to buy *"... used SR-51s and SR-56s for resale to my students at Manhattan Community College at cost."*

Oil and Gas Well Accounting and Taxes: L H Southmayd (1097) would like to make contact with other members interested in programming in these applications areas.

PC Battery Charging: Several members have noted that the PC-100A can easily overcharge an installed battery pack, since charging occurs even with the power switch off. This problem was covered in V2N5p3, along with the suggestion that members wishing to try hardware mods to provide more charging control contact Bob Edelen (100).

Games Buffs: Walter Fair (1056) enjoys programming games, and invites other interested members to contact him. Incidently, members with specific applications interests/problems may find it worth looking through back issues of 52-NOTES to identify likely correspondents from their contributed material. Be sure you are up to date with membership address changes before writing.

PC Roller Cleaning: The question came up recently, but TI has no specific recommendation for cleaning the rubber roller in the paper drive mechanism. A typewriter roller-cleaning solvent should do, but in any case, don't let the solvent contact the printheads.

More on Op3mn on 2-Digit Registers (V3N9p5**):** Bill Skillman (710) and Maurice Swinnen (779) note that the reg 40 exception (V2N12p2) applies. Maurice further notes that Dsz*mn ab works for all registers, as expected (V2N7p6).

CROM Calls to Undefined Labels: Steve Bepko (45) notes that at turn-on, a call to ML-09 D puts the pointer "out of bounds", such that LRN key cycling will not switch to LRN mode. It appears that the ... Pgm 00 A' ... sequence at steps 062-064 causes the hangup, since A' is undefined in Pgm 00 (user memory). But as Steve points out, a call to a similar sequence in ML-08 (Pgm 8 E) doesn't disable the LRN function, and neither does a call directly to Pgm 9 SBR 062 or PGM 8 SBR 055. It seems that the ML-09 call to its own routine E at step 061 before trying to call Pgm 00 A' makes the difference, but why? (Before trying Pgm 8 E, initialize Reg 2 so its contents is greater than that of Reg 1, to avoid invoking a confusing overflow signal).

Membership Address Changes: 544: 863 Post Ave Rochester, NY 14169; 938: 3418 El Potrero Dr Bakersfield CA 93304; 973: 81 Stanton Rd Brookline, MA 02146.

Volume 3 Number 12 48/39/38 December 1978

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

Roots of Polynomials (V3N9p1)

Dix Fulton (83) and Frank Stallings (389) join Bill in addressing this topic. Some time ago Dix had submitted a Quartic, Cubic, Quadratic program to PPX-59 (#338005B), and has sent me an improved version, listed below. This is shorter than Bill's V3N9 program, primarily because the same processing is used for all 3 degrees: All 5 coefficients are entered, regardless of the degree. For a quadratic, D and E are zero, and for a cubic. E is zero, since the "quartic" $Ax^4+Bx^3+Cx^2+0x+0=0$ can be divided by x^2, making it the quadratic $Ax^2+Bx+C=0$ which it really is, and similarly the "quartic" $Ax^4+Bx^3+ Cx^2+Dx+0=0$ reduces to a cubic. But quadratic and cubic processing are no faster than quartic, and the correct 2 or 3 roots must be identified among the always output 4.

Dix spotted a flaw in Bill's V3N9 program: In same cases precision is unnecessarily lost following a rounding procedure. Bill has corrected that, and reworked quadratic processing to improve accuracy when the 2 discriminant terms have a large difference in magnitude (V3N9p2). So for some cases, Bill's new program, listed below, is more accurate than Dix's.

Maurice Swinnen (779) has translated for English-speaking 59/PC users a Quartic, Cubic, Quadratic program written for German-speaking users, appearing in **Display** (V3N4/5S91). However, it requires 4 card sides of memory, and suffers the quadratic accuracy loss with critical discriminants.

Frank Stallings has been experimenting with iterative methods for finding the roots of higher order polynomials, and wrote the program listed below following the so-called Bairstow method. Frank's program appears to work well on a few sample problems, producing both real and complex roots for polynomials up to 61st degree, at a rate of about a minute per degree. However, there are no known iterative methods which work for all polynomials, and the Bairstow method (like the Newton method which it uses) is known to be bad for multiple roots solutions. So in cases where convergence is slow, Frank has provided a flag option allowing the monitoring of successive root approximations.

TI-59/PC Program: Polynomial Roots L Frank Stallings (389)

User Instructions:

Press RST, R/S; see the prompter: DEGRE printed. Key degree (2-61), press R/S; key the coefficients, beginning with the highest, and follow each with R/S. Processing begins following input of the constant. Monitor convergence (or lack thereof) by setting flag 0. With good convergence, roots are printed

in pairs about every 2 minutes.

Program Listing:

```
000:  CMs Adv Adv Op00 1617223517 Op2 CLR Op05 7 Op17 R/S Prt Adv GTO
027:  053 Lbl E EE INV EE rtn Lbl B 1) (RCL*69 ÷ RCL0 rtn Prt STO*69 1 SUM69
      +/-
053:  SUM68 RCL68 CP x≥t 045 Lbl C Adv Adv 3532323736 Op2 Op5 GTO 084 Prt
083:  Adv Dsz69 p51 RCL*69 x=t 082 2 +/- ÷ RCL69 ÷ RCL0 X (CP X RCL1) STO62
      RCL2 ÷
112:  (1 +/- ÷ RCL69 = STO64 RCL64 X RCL68 ÷ RCL63 X RCL67 + RCL*62) (Exc67
      STO68 RCL64
143:  X RCL66 ÷ RCL63 X RCL65 + (1 SUM62 INV SUM69 R69 CP x=t 121) INV x≥t
169:  181 RCL67) Exc65 STO66 GTO 121 SUM62) 0 = X STO69 RCL66 +/- + RCL65 x² =
197:  Exc66 X RCL67 - 0 Exc68 X RCL65 = ÷ 0 Exc66 INV x=t 219 B = SUM63
222:  Prd69 Abs + ((0 Exc62 Exc69 ÷ 0 Exc67) ÷ 0 Exc65 INV x=t 247 B)
248:  INV SUM64 Abs = X 1 EE 8 = Op0 INV Ifflg0 266 Pause INV x=t 121
270:  CLR 2 INV SUM69 1/x X (RCL63 x² + 4 X RCL64) INV x≥t 308 (√x + RCL63)
297:  - E Prt Adv RCL63 = E GTO 331 +/- √x = E x:t 24 Op4 RCL63 ÷ 2 = E Prt
324:  x:t Op6 Adv x:t Prt x:t +/- Op6 Adv RCL*62 X (CE X 1 SUM62 RCL62 x:t
347:  RCL69 INV x≥t 370 RCL63) SUM*62 1 SUM62 RCL64 = SUM*62 Dsz62 335 CLR
371:  STO62 2 x:t RCL69 x≥t 121 x:t CLR x=t 395 RCL1 ÷ RCL0) +/- E Prt Adv
      RST
```

Note: The p51 at step 086 is the pseudo BST which must be created synthetically (R51 BST BST Del)

TI-59/(PC) Program: Quadratic,Cubic,Quartic Roots Skillman(710)

User Instructions: Same as V3N9p2

Program Listing:

```
000:  GTO 620 Lbl E' SBR623 rtn Lbl D' 1 Exc1 1/x Prd2 Prd3 Prd4 rtn Lbl B'
      D'
025:  35 Op4 RCL2 ÷ 3 = STO8 x² +/- + RCL3 ÷ x:t 3 = STO10 RCL4 + RCL8 X (x² X
      2 -
058:  0 x:t = STO11 x² + RCL10 X x² X 4 = INV x≥t 147 √x + x:t R11 = ÷
083:  2 +/- + SBR126 x:t = SBR126 + x:t - RCL8 + Ifflg2 142 E' RCL8 = ÷ 2
110:  +/- + x:t = X 3 √x = x:t - RCL8 = GTO 609 (STO13 Op10 Exc13 Abs INV yˣ
136:  3 X RCL13) rtn 0 = STO21 rtn RCL10 +/- √x X x:t 2 = STO12 RCL11 ÷ 2 ÷
      x:t yˣ
164:  3 = +/- Rad INV Cos ÷ 3 = STO9 SBR196 STO21 SBR187 STO22 2 X 2 X π ÷ 3 +
193:  RCL9 = Cos X RCL12 - RCL8 = STO23 Ifflg2 719 GTO E' Lbl A x:t 3 Op17
      CMs
219:  x:t STO1 x:t 13 RST Lbl B STO2 x:t 14 RST Lbl C STO3 x:t 15 RST Lbl D
      STO4 x:t
247:  16 RST Lbl E STO5 x:t 17 RST Lbl A' D' 35 Op4 RCL2 ÷ 2 = +/- STO6 STO7
      x² – RCL3
279:  = CP x≥t 296 +/- √x x:t RCL6 Ifflg4 516 GTO 609 √x X RCL6 Op10 =
303:  SUM6 RCL3 ÷ RCL6 GTO 628 Lbl C' RCL2 CP x=t 469 D' Prd5 RCL2 X RCL4 – 4
      X RCL5
334:  = Exc3 STO15 X +/- Exc2 STO14 4 - RCL14 x² = X RCL5 - RCL4 x² = Exc4
      STO16
363:  Stflg2 20 STO0 SBR025 INV Stflg2 CP CLR Op20 RCL*0 + RCL14 x² ÷ 4 -
388:  RCL15 = SBR581 INV x≥t 376 √x STO19 +/- + RCL14 ÷ 2 INV Prd*0 = STO2
      RCL14
415:  X RCL*0 - R16 = Op10 X (RC*0 x² - R5) SBR581 INC x≥t 376 √x = STO20
444:  +/- + RCL*0 = STO3 SBR261 RCL19 SUM2 SUM2 RCL20 SUM3 SUM3 GTO 261 RCL4
      INV
472:  x=t 321 RCL3 x=t 540 RCL5 Exc3 STO2 Stflg4 GTO 260 SBR496 RCL7 CP x≥t
498:  511 +/- √x x:t SBR616 +/- x:t GTO 616 √x E' +/- GTO E' x:t INV P/R ÷ 2
521:  = x:t √x x:t P/R x:t STO6 SBR 603 x:t 1 +/- PRD6 GTO 603 RCL5 ÷ RCL1 =
546:  x≥t 569 +/- √x √x x:t 35 SBR620 +/- E' +/- x:t SBR616 +/- GTO E' Nop ÷
570:  4 = √x √x X x:t 1 = GTO 527 Nop STO18 Fix4 EE INV EE INV Fix INV
592:  x=t 597 CLR rtn RCL18 rtn Nop Nop Nop 35 Op4 RCL6 E' 47632024 Stflg3
620:  Op4 x:t Op6 GTO 705 = STO7 RCL6 Ifflg4 491 E' RCL7 GTO 623
705:  Op8 Ifflg3 717 Op69 INV Stflg3 R/S CE rtn
```

TI-59 Program: Quartic,Cubic,Quadratic Solutions Dix Fulton(83)

User Instructions:

Input coefficients per V3N9p2, following a manual CMs. Press E' and see 4 roots printed; or without printer press E', see 0, press A', see root 1, press B', see root 2, press C', see root 3, press D', see root 4. For each root, the imaginary part is pause-displayed, followed by an x:t displaying the real part.

Program Listing:

```
000:  Lbl A' INV Lbl B' Stflg1 6 GTO STO Lbl C' INV Lbl D' Stflg1 8 LBL STO
      STO9
022:  RCL*9 ÷ 2 = x² - Op39 RCL*9 Op29 = CP x≥t Rcl Abs √x x:t Lbl Rcl
044:  √x Ifflg1 SUM x:t +/- x:t +/- Lbl SUM - RCL*9 ÷ 35171327 Op04 2 = Adv
071:  Op6 x:t ST09 24301322 Op4 RCL9 Op6 Pause x:t rtn Lbl E' RCL4 1/x Prd0
100:  Prd1 Prd2 Prd3 RCL1 STO6 x² + RCL3 Prd6 x² X RCL0 + RCL2 +/- STO7 X (4 X
      RCL0)
133:  INV SUM6 = +/- STO5 RCL7 ÷ 3 X STO7 RCL6 - RCL5 = ÷ 2 - RCL7 X x² = STO8
      x² +
165:  (RCL6 ÷ 3 - RCL7 x²) X x² = CP x≥t Cos +/- √x x:t RCL8 x:t INV P/R
189:  x:t INV yˣ 3 = x:t ÷ 3 = P/R X 3 √x ÷ 2 = STO6 +/- STO5 x:t INV SUM5
214:  INV SUM6 X 2 = x:t RCL5 INV x≥t Sin x:t Lbl Sin RLC6 x≥t Tan x:t
234:  GTO Tan Lbl Cos √x SUM8 +/- X 2 + RCL8 + x:t = Op10 X x:t Abs INV
256:  yˣ 3 + RCL8 Op10 X RCL8 Abs INV yˣ 3 Lbl Tan - RCL7 = STO6 ÷ 2 = STO5 +
284:  ((x² - RCL0 + Abs) ÷ 2) √x INV SUM5 = STO7 RCL3 ÷ 2 = STO8 + (x² + R6
316:  - RCL2) √x INV SUM8 = STO6 X RCL7 + RCL8 X RCL5 - RCL1 = Abs x:t RCL6
      X RCL5 +
348:  RCL8 X RCL7 - RCL1 = Abs x≥t Exc RCL5 Exc7 STO5 Lbl Exc RCL4 Prd0 Prd1
      Prd2
376:  Prd3 0 HIR8 1 Op4 HIR18 CP x=t Prd INV Fix A' B' C' D' Adv Adv Adv
398:  Lbl Prd R/S Lbl A STO4 x:t 1305 Op4 x:t Op6 R/S Lbl B  STO3 x:t 1404
      Op4
427:  x:t Op6 R/S Lbl C STO2 x:t 1303 Op4 x:t Op6 R/S Lbl D STO1 x:t 1302 Op4
457:  x:t Op6 R/S Lbl E STO0 x:t 1301 Op4 x:t Op6 R/S GTO E'
```

The Math/Utilities CROM (V3N8p5,6)

A few pre-production modules have now been made, and TI expects production to begin mid-December, with first deliveries expected by early January. Many of the programs in this new CROM do indeed reflect better programming quality than has gone into earlier ones, and the routines themselves will probably turn out to be the most universally useful. Fortunately, subroutine callability (from RAM) was one of the design criteria. Following is a brief run-down of the 21 programs:

MU-01 is the usual module check and register-clearer. (60 steps)

MU-02 works only with the PC, providing a few common prompting messages, including multiple-card read-write directions with code which cleverly interrogates the first card-side read to determine how many more sides are to be read. Unfortunately, the prompter refers to "cards" instead of "sides", which may confuse some users. (329 steps)

MU-03 is similar to the LE-10 Memopad, but with more symbol-code keyboard-addressable, and with the words ENTER, PRESS, and PRESS SBR single-key specifiable. Users will need to make their own overlays for finding the alpha prefixes, as TI budget constraints did not allow for one to be included in the module package. (474 steps)

MU-04 is used with MU-03 to format both data and text in any desired configuration for each line. It's slow, but can save paper. (430 steps)

MU-05 is titled Superplotter, and makes it easy for the user to generate plots of up to ten different functions, with size and precision limited only by acceptable execution time and paper. The result is one long tape which the user cuts and reassembles with the aid of wellchosen alignment and coordinate symbols.
This looks like a winner.(444 steps)

MU-06 is the fast sorter (V3N8p6) which orders one sequence of 99 random numbers in 9¼ minutes, but which takes 19½ minutes for the same code executed in RAM. The V3N2p5 program, which was TI's point of departure, takes 23 1/3 minutes to order the same sequence. The MU-06 interface with the user was well planned, giving the user a good choice of I/O options for either keyboard or RAM program call. (132 steps)

MU-07 is a data array processor, and at 650 steps is the longest MU program. Up to 93 (53 for the 58) elements of a 2-dimensional array can be arithmetically combined or changed by row or column, and the elements shifted right or left by row. There are a lot of business oriented manipulative options, probably best run via a RAM program with lots of prompting. MU-07 appears to be an extension of the Business Decisions Project Planning and Budgeting program (BD-05).

MU-08 is a general-purpose data packer and unpacker of positive integers, allowing the user to specify mixed as well as constant data lengths for 13-digit per register packing into Registers 4 onward to the end of partitioning. The 3 functions: store, recall, and exchange each take about 4 seconds to execute, and may be invoked in any order any number of times. (226 steps)

MU-09 is a disappointing factor finder: It's short (TI claims it needed to cut down on memory requirements, although MU-09 is only 7 steps shorter than the V3N11p4 program), but slow (doesn't bypass trial divisors which are multiples of smaller ones already tried (V3N11p4), and does quite a bit of flag manipulating). (85 steps)

MU-10 is a fast, straightforward generator of hyperbolic trig functions, and their inverses. (100 steps)

MU-11 calculates the gamma function and factorials for both integers and positive reals up to 69 or 70, and the natural logs of these 2 functions for reals up to about 10^{10}. (147 steps)

MU-12 is a short random number generator which produces uniformly distributed numbers in the 0-1 range the same way ML-15 does, but a bit more efficiently. It uses MU-13 to generate normally distributed numbers. (54 steps)

MU-13 does normal distribution processing, 2 of the functions producing the same results as 2 of the Applied Statistics Normal Distribution program (ST-19); Routines A and C of MU-13 correspond to routines C and A, respectively, of ST-19. (260 steps)

MU-14 mechanizes the Aitken interpolation algorithm, fitting up to a 26th order polynomial to 27 data points (13th order for 14 data points for the 58), which is then used to solve for intermediate points. (193 steps)

MU-15 is a primitive root-finder, requiring the user to provide approximations to each (real only) root. It does provide an option to specify the max number of iterations for cases where convergence is expected to be slow or maybe nonexistent. (99 steps)

MU-16 finds the max and min values of user-supplied functions in specifiable intervals by looking for a change in sign of an approximation to the function's derivative. This approximation (also used by MU-15) imposes processing limitations of which the user needs to be aware. It would seem that a straight forward sample-and-save-largest-and-smallest approach (V2N8p4,5) would have been better. (211 steps)

MU-17 mechanizes Romberg integration of definite integrals, allowing the user to specify accuracy. (243 steps)

MU-18 uses a 4th-order Runge-Kutta method to solve differential equations of the y'=f(x,y) and y''=f(x,y,y') types. (293 steps)

MU-19 performs discrete Fourier series summations on as many input equally spaced data points as there are registers available from Reg 16 onward. If a user's RAM program needs a few registers from Reg 16 on, he can specify a higher start register for MU-19 to use for input storage. (115 steps)

MU-20 does a good job of determining, recording, and resetting the status (V3N2p2) of flags, partitioning, angle mode, and Fix. The number of open parentheses, and printer connection can be determined, but not recorded or reset. (307 steps)

MU-21 is designed to aid manual (keyboard) calculations by assigning 5 of the user-defined keys to specified variables and subroutines. I can see where single-key variable recall might help to speed things up for the non-programmer, but since the user has to write whatever subroutines he wants, anyway, he might as well call them directly. (101 steps)

I'll comment further in future articles, as I delve into the MU programs in greater detail, and/or start getting inputs from the membership.

Revealed Firmware (V3N10p4**,** V3N11p1,2**)**

Steve Bepko (45), Maurice Swinnen (779), Dave Leising (890), and John Mickelsen (990) have all found more ROM code past step 487. Steve and John got to step 575 by single stepping without the printer; Maurice and Dave to step 583 with the printer in trace mode. Apparently no one else had bothered to SST past step 487, since what Steve and John did works with any of the established ROM-revelation procedures. For a shortcut confirmation, key at turn-on: GTO 479 9 Op17 Pgm 12 SBR 444 R/S P/R LRN, and SST to step 575, at which point the next SST causes a switch to RUN mode.If LRN is then pressed, the whole keyboard appears to be locked out with a displayed C.

Confirmation of Maurice's and Dave's approach can be made with a 59/PC at turn-on with: Stflg 9 9 Op17 Pgm12 SBR 444 P/R P/R LRN BST which lists steps 000-583 without mnemonics, and can only be stopped by turning the machine off. If instead of keying the Stflg 9 you press (latch) the TRACE key, printing will stop when the TRACE key is unlatched.

Steve found that some regular (with mnemonics) listing could be made past step 487 by first SSTing past step 487, then keying LRN List. It turns out that step 489 is the lowest that will work, and in any case step 503 is as high as it goes, starting over at step 039 set to code 80 (Grd) this time, and continuing on listing RAM from step 040 to the end of the current partition.

Bill Skillman (710) identifies steps 384-511 as non-normalized constants used by the transcendental functions. (Steps 512-575 repeat steps 384-447). Bill finds that if the significant digits in the equivalent 16 data registers are allowed to be rescaled, they can be interpreted as: ln 10, ln 2, ln 1.1, ln 1.01, ln 1.001, ln 1.0001, ln 1.00001, ln 1.000001, π/4, arctan .1, arctan .01, arctan .001, arctan .0001, π/2, π, and 180/π. If steps 384-447 are written in RAM, and recalled as data, these 16 constants appear as: -ln 10 X 10, ln 2 X 10^{-94}, ln 1.1 X 10^{-32}, ln 1.01 X 10^{102}, ln 1.001 X 10^{-8}, -ln 1.0001 X 10^{33}, ln 1.00001 (no change), ln 1.000001 X 10^{50}, π/4 X 10^{45}, arctan .1 X 10^{20}, arctan .01 X 10^{67}, -arctan .001 X 10^{-66}, -arctan .0001 X 10^{-66}, -π/2 X 10^{2}, -π X 10, and -180/π X 10.

While the revelation of these constants raises new questions as to how data are formatted during transcendental function processing, their identity and precision should help in the determination of the algorithms used (V2N9p6).

Dave reports that the ROM resides *"... in the TMC 0571 chip located on the lower right corner of the PC board. This chip is essentially a 'CROM' intended for the storage of resident..."* built-in functions.

Two New Periodicals

Ken and Jon Mills, who used to write a Recreational Programmer column in **65-Notes**, have begun publishing an independent bimonthly magazine of the same title. V1N1 is dated Sept-Oct 78, and addresses an assortment of calculator-computer topics including business programming, transcendental functions, celestial navigation computations, a couple of games, PPC cryptography, and a book review. V1N2 continues the business programming column and another book review, adding several more games. so far, most of the material is HP PPC oriented, but the fairly generous flow charting and English descriptions accompanying program listings should facilitate translation to other machines. Contributing authors come across as enthusiastic, writing in an easy style, though at times somewhat rambling and short on rigor. Subscription is $12 per year in the US; $15 elsewhere. For more information write to: The Recreational Programmer Box 2571 Kalamazoo, MI 49003.

Didactic Programming, A Journal of Calculator-Demonstrated Math Instruction, began with a Fall 1978 issue aimed at math instructors who are integrating PPCs into their classrooms. Topics range over iterative equation solving, Fibonacci Search for relative minima, Gaussian Elimination, and a 3 simultaneous equations solver. Continuation and periodicity of this publication will be determined largely by the amount of contributed material received. For further information, write to Didactic Programming Box 974 Laguna beach, CA 92625.

Tips and Miscellany

Membership Address Changes: 343: RR 1 Box 176 Blue Earth, MN 56013; 606: 1740 N Cherry St Mesa, AZ 85201; 1031: 2141 Steiger Ln Oceanside, CA 92054; 1072: 18082 47th St Brooklyn, NY 11204.

The Washington (DC) Area Local Club: The only organized TI PPC local group I am aware of is the one Dave Johnston (5) and Maurice Swinnen (779) started (V3N4p5). Maurice reports a current regular membership of 15, and some Friendly Competition with a local HP PPC group. A current challenge is to use only the functions: = (or ENTER), x:t (or x:y, or Exc 00), y^x, 1/x, $\sqrt{x}$, x^2, and INV lnx (or e^x) on a PPC at turn-on, and display a 3 with the fewest steps, where INV lnx and Exc 00 each count as only one step. No other key may be used, including the numerals and CLR. Maurice has a 14-step solution, which I've trimmed to 11. Send your best to Maurice or me. It should be interesting to see whether a TI or HP PPC is the ultimate winner. The Washington area members have been noteworthy contributors to 52-NOTES, both individually and collectively, and I know I speak for all of us when I hope they keep up their productive activities.

Volume 4 Number 1 48/39/38 January 1979

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

Off-the-Shelf Custom CROMs

A number of companies have commissioned TI to produce CROMs to their specifications, in some cases with the intent of marketing them as part of special-purpose 59/PC packages for small business and professional groups. Although the vendors of such packages obviously prefer to sell them in as comprehensive a configuration as they can, including software support and hardware maintenance, they may be willing in some cases to sell the modules separately to users who have their own machines and who need no help either using the custom CROMs directly or in writing RAM programs to call CROM routines.

From what I've learned sofar from people who have bought custom CROMs, code is apt to be protected (can't be downloaded), and a single module could cost a second buyer $600 or so. This, of course, includes a share of the software engineering which goes into its design and checkout, as well as TI's charges for emulator time (typically $2000) and fabrication, and vendor profit. This compares with going to TI directly with your own programs, and paying $12,000 for 250 units, $17,000 for 500, or $25,000 for 1000.

Anyway, here are a few facts, figures and contacts for members who may wish to take advantage of already-made custom CROMs. I make no representation whatsoever as to quality or utility, but merely pass this information along, hoping it will be useful to some members.

Reynolds & Reynolds, Dayton, OH 45401 (513) 226-0808 x539 is marketing a 59/PC custom CROM package which they call the COMPEG 2001 microprocessor for payroll computations. TI identifiers have been covered over by R&R markings, and a colored keyboard overlay labels special-use keys. The system is designed to automate so-called pegboard payroll accounting, with one mag card *"... containing the entire payroll history of one employee."* The total package costs $1300, plus $100 per year Maintenance fee.

Horizons Technology, 7830 Clairmont Mesa Blvd San Diego, CA 92111 (714) 292-8331 *"... currently does custom programming for government agencies. We design and program custom CROMs, produce our own magnetic cards, and distribute comprehensive software documents. Most of our customers are given a large system that is cost-effective; small individualized programming jobs would have to be assessed on a case by case basis."*

Z-Comp Division of Davis-Gilbert Co, Inc #616 The Landmark Bldg Alamo Plaza, San Antonio, TX 78205 (512) 227-7777 has a Texas-oriented custom CROM covering

consumer installment financing loan computations, aimed at banks as primary users. Prices are $970 with 59/PC, $835 with 59 alone, $795 with 58/PC, $660 with 58 alone, and $575 for the CROM only. This company plans to have other modules made which would apply specifically to other states.

Other custom CROMs include: Oil and gas drilling engineering computations by IMCO Services Box 22605 Houston, TX 77027; and Ophthal- mological computations by Sonometrics, Inc 16 W 61st St NY, NY 10021. Members with additional information concerning either the availability or quality of custom CROMs are invited to share same.

The Game of Life

A decade or so ago, University of Cambridge (England) mathematician John Horton Conway developed a solitaire game which he called LIFE, having as its purpose the challenge to the player to accurately predict successive changes to initial starting patterns resulting from the application of a few simple rules, and to formulate behavioral hypotheses to support the predictions. As originally conceived, LIFE also challenges the player to avoid easy-to-make playboard mistakes! The latter can be, and has been eliminated by computer mechanization, giving way to a new challenge: Writing efficient programs to produce any desired number of successive patterns from input initial patterns. It is this challenge which will be addresses in 52-NOTES. Members wishing to pursue the playing challenges are referred to Martin Gardner's Mathematical games discussions, beginning with the October 1970 issue of **Scientific American**.

The programming requirement is to take an input 2-dimensional array of elements (organisms) and produce successive new arrays, each derived from its predecessor according to 3 rules: 1) An existing element survives it is has 2 or 3 neighbors, 2) A new element is created (born) when an empty position has exactly 3 neighbors, and 3) An element vanishes (dies) when condition 1) is not met. A neighbor is defined as an existing element occupying any of 8 possible positions surrounding a given position, and all changes are considered to occur simultaneously. Conceptually, the 2-dimensional field is infinitely large, to allow patterns to move/grow without restriction. Practically, of course, a physical board, or computer memory is of finite size, imposing an upper limit on pattern movement or growth.

A practical 59/PC limit would appear to be a 20 X 20 field, within which a reasonable amount of movement/growth can be accommodated by restricting input patterns to the center 10 X 10 positions. Over the past year or so, I've received 59/PC LIFE programs from Mike Brown (128), Rusty Wright (581), Lou Cargile (625), Bill Skillman (710), and Maurice Swinnen (779), the latter being a translation of a German version appearing in **Display** (V4N1/2) written by a Peter Klinghardt, whose program inspired Bill to write his. Of these, I judge Bill's to be the best, and his 13 Nov 78 version lists below. Bill does some tricky things, and it's a challenge just to figure out how his program works! He chose the - to represent an empty position, the 0 (zero) for an established organism, a Q for a new birth, and a blank to signify a recent death. While Bill's choices are good, they may not be the best for all users, and members wishing to experiment with other characters can do so by altering the numerals at steps 147-148, 159-160, and 162-163, which as written produce the blank, 0, and Q respectively. For example writing 50 at steps 159-160, and 04 at steps 162-163 changes the 0 to II and Q to I.

TI-59/PC Program: Life **Bill Skillman (710)**

User Instructions:

Key g.i where g is the generation number (0 or 1 if just starting) and i is any desired identifier, press E. Input the starting pattern as keyed strings of zeros and ones, up to ten per line, press R/S following each input, see next line number displayed. Processing begins automatically following input of the tenth line, or may be initiated sooner by pressing A.

Successive generations are printed indefinitely until stopped with R/S. To prepare to continue a run from one turn-on to another, push R/S immediately following the printing of the next population sum (s Σ), note the current step number, and record banks 3 and 4. At next turn-on, read the recorded card into banks 3 and 4 (in addition to a program card into banks 1 and 2), and press GTO 222 R/S, unless the last stopped step was between 216-220, in which case press GTO 220 R/S

Program Listing:

```
000:  Lbl C' = SUM 55 Lbl E' INV Int X 1 EE 10 = INV EE rtn Lbl B' RCL 56 Lbl
      D INV EE
026:  Op*01 Op 31 rtn Lbl D' Op 24 RCL*04 STO 59 0 STO 53 RCL 54 ÷ 1 EE 5 =
      INV Int x:t
052:  0 INV x≥t 069 B' 1 EE 5 +/- Prd 54 Prd 59 GTO 072 SBR 086 3 x:t RCL 54
076:  x≥t 341 B' 1 EE 4 GTO 347 RCL 56 ST0 08 5 STO 09 CP RCL 54 Int X .1
      Prd 53 =
103:  STO 54 x=t 172 RCL 59 Int X .1 = STO 59 INV Int INV x=t 124 INV Stflg 2
      NOP
127:  RCL 54 INV Int x:t .3 x=t 154 INV Ifflg2 172 .4 x=t 159 60 SUM8 GTO
152:  172 INV Ifflg 2 162 47 + 14 = SUM08 1 SUM53 Op20 .01 Prd8 Dsz9 093
181:  RCL8 SBR 009 D rtn 221721 Op4 RCL2 Op6 Nop rtn CLR 77 Op04 0 STO8 STO9
212:  Exc0 Op6 Adv CP x=t R/S Op 22 SBR188 20 STO3 7 STO5 9 STO4 ST06 11 STO7
241:  CP SBR 325 STO 54 x=t 262 ÷ 10 ÷ SUM54 1 EE 8 C' SUM54 SBR325 x=t 284
268:  STO 53 SUM 55 X 10 ÷ SUM55 100 C' SUM54 4 STO1 D' Exc58 OP35 STO*5 Op25
296:  0 Exc 55 STO 54 D' Op 05 Exc 57 STO*05 Dsz 3 241 0 Exc 58 STO 48 0 Exc
      57 STO 49 GTO
323:  202 Op 25 Op 26 Op 27 RCL*05 + RCL*06 + RCL*07 = rtn SBR 086 1 EE 9 X
      RCL53 =
351:  rtn Lbl E CMs STO2 SBR188 2020202020 STO56 20 STO1 10 STO7 11 - RCL7 =
      R/S
387:  . Prt ÷ 1 EE 5 X 3 = STO*1 E' Exc*1 Int Op21 STO*1 Op21 Dsz7 380 Lbl A
414:  SBR 446 10 STO 07 20 S06 4 STO1 SBR 463 SBR 463 Op 05 Dsz 7 425 SBR 446
      GTO
444:  202 5 STO 05 4 STO 01 B' D D D Op 05 Dsz 5 456 rtn RCL*06 STO 54 2 STO
      04 D' STO*06
473:  Op26 rtn
```

Note: Bill's program was written with Adv at step 200. Also note that the . at step 387 is to clear a hardened display for all-zero inputs. Members are invited to try to out-do Bill with better (faster, more versatile, etc) LIFE programs.

More on Program Copyright (V3N5p4,5)

Ken Widelitz explores the topic further (**Kilobaud** Jan 79 pp 9,10), reviewing the National Commission on New Technological Uses of Copyrighted Works (CONTU) final report, and its recommendations to Congress for amendments to the Copyright Act of 1976. While Widelitz states that he agrees with the conclusions of the report, much of the text he airs leaves me with the impression that for all of its attempted specificity, the CONTU report fails to provide an achievable means to define, much less protect the rights of originators of computer readable works.

My own feeling is that the only practical way to prevent program plagiarism is to deny the type of access to the program that makes copying possible. But that's easier said than done. Users have already succeeded in getting at and copying the 59 code on a "protected" card, and I suppose there might be some effort being devoted to revealing protected CROM code. However, the CROMs present an interesting special case: revelation of their code may actually enhance their marketability, since potential users would be able to verify what they're getting, and could very well find that buying the CROM (TI or private custom off-the-shelf) beats filling up RAM with plagiarized slower-executing code.

Nevertheless, TI considers its CROMs to be copyrighted, even though it doesn't publish listings. The label on a CROM does indeed contain the circled c and the year, but perhaps that could be construed to apply only to the label itself. The question of whether programs cast in ROM are patentable or copyrightable appears not to have been conclusively answered yet, since as Widelitz states: *"Some companies have successfully patented computer programs by getting the patent on 'firmware'. Since patent and copyright are mutually exclusive, it will be interesting to see how this conflict is resolved."*

Fred Fish's User Survival Guide (V3N3p1) may turn out to be an interesting test case, since it is copyrighted, and lists the ML CROM programs. Who knows, it might turn out that TI would be required to obtain Fred's permission to publish its own listings!

Book Review: Sourcebook for Programmable Calculators, TI Learning Center, 1978, $12.95, 416 pages

This TI publication was designed to provide the user of 58/59 machines with a wide variety of straight-forward applications programs, each accompanied by a brief tutorial. Topics span number theory, algebra, trig, calculus, statistics, music, business, economics, biology, medicine, engineering, physics, and astronomy, with pedagogical treatment generally as light as is possible for each subject. No programming experience is assumed, and program understandability appears to have dominated over I/O and execution efficiency considerations when the routines were designed.

While there is no attempt to teach programming techniques, each listing and corresponding user instructions include Purpose/Comments, and Display/Comments, respectively, facilitating user comprehension. Master Library routines are called when appropriate, but there are no warnings to the user to assure ML module installation.

I haven't yet checked out all the algorithms and routines, but they appear to have the backing of a panel of college-level educators, and I recommend this book to TI PPC users who need help getting started at mechanizing algorithms on their machines, and to users of computing machinery in general who are looking for brief introductions to the covered topics. Users are, of course, advised to be on the lookout for errors. For example, Jerry Johnston (493) spotted some questionable code, and suggests that 2 STO instructions at locations 042 and 061 of the equation of Time Routine (page 11-39) should be deleted. I suspect that the STOs in question should be replaced by GTOs.

CROM RNG Limitations

Don Huffman (902) has found that TI's random number generator used in several of the CROMs doesn't always guarantee a period of m (199017, as stated on page 54 of the ML Manual). In fact, a bit of trial and error turned up 2 unrelated 13-digit seeds which Don found produced sequences with periods under 500, which means that a little search produced over 900 seeds which would initiate sequences of cycle length less than 500.

It turns out that with the linear congruential method, a minimum cycle of length m is only guaranteed when the seed is an integer and subsequent seeds are carried as exact reals (see Theorem A in Knuth's **"The Art of Computer Programming"** Vol 2 p 15, and Exercise 1 p 31).

But it may be that TI's routine carries most successive reals to sufficient precision to meet the exact reals requirement, provided the input seed is an integer. Has anyone found an integer seed which produces a cycle shorter than m?

The PC-100C Printer

TI is currently marketing domestically a printer labeled PC-100C, which appears to be similar to the 100B (V3N9p6). The 100C differs from the 100A primarily by being adaptable only to the 58/59 machines, having a non-detachable power cord, and allowing calculator operation with PC power off.

A 100C which I've tried is noisier than other PCs I've used, a characteristic which may just be an isolated case. Calculator operation with PC power off is the same as when on, as far as PC connection sensing is concerned (i.e. Ops 1-4 load print buffers, Op 7 on a display greater than 19 causes an error condition, and Op 8 initiates a label search to the end of the current partition (but without printing labels)). The 3% execution slowdown (V3N7p6) appears to prevail with PC power either on or off.

Externally, the 100C looks about the same as the other PCs, but a look at the inside, with the top removed, shows quite a few changes. The PC board (here, PC means printed circuit, not print cradle!) shows marked differences in circuitry and components. What appear to be 2 high-power diodes are fastened to 2 much smaller heat sinks, eliminating the 100A's one large one, and a single large capacitor has been replaced by 2 smaller ones. The print assembly appears to be about the same, although the step-motor and transmission are different in appearance, and made by a different manufacturer. (Maybe this accounts for the higher print-noise level). Some components on the PC board nearest the tape supply have been relocated, and a cardboard guard eliminated.

Further comparisons among the various PC-100 models are invited from the membership.

A CROM Op Ind

Dave Leising (890) has found an ML-19 point (SBR 172) to execute an effective Op*53, making it possible to have any of the 40 Op functions executed in the CROM. One that he found interesting with printer connection is Op 8, which produces a somewhat odd mnemonicless listing of RAM code. To see what happens, at turn-on, key 10 Op 17 8 STO 53, write some code at both ends of the partition, press (latch) the TRACE key, and key Pgm 19 SBR 172. The first 2 steps are printed as 000 01 and 001 00 followed by the "real" steps 001-158. Step 159 is printed as a code 43 (RCL), and execution halts following an effective Prt of the 8 in the display. Inspection of the last step confirms the code 43, but a look at step 000 shows a code 85 (+). If during the listing the TRACE key is unlatched, label-search appears to continue without further printing (even labels are ignored). If the TRACE key is again latched before the search ends, printing resumes with the remaining steps.

While the CROM Op*53 executing as Op 8 causes execution to halt at the end of the label search, other Ops let the CROM code continue to execute, resulting in error setting and endless looping, unless several key registers have been appropriately initialized. So be prepared to key RST to halt execution, when necessary. Members are invited to examine the behavior of other CROM-executed Ops, an to share their findings.

Tips and Miscellany

Printing 12 Places (58/59/PC): Mike Brown (128) has discovered an efficient way to get 12 of the 13 mantissa places printed. He found that in TRACE mode, a RCL Ind on a register whose contents is a 13- place integer causes 12 of those places to be printed! (albeit along with a little garbage and an error condition). Invoke the following sequence by pressing A, with the desired number to be printed in the display register: *000: R/S **Lbl A** Prt STO 00 1 EE 50 Prd0 CLR Stflg9 RCL*0 CLR RST* and see the number as displayed printed, followed by a trace or RCL*0, which includes the unscaled 12 MSDs of the input number. The 1 EE 50 is to assure that the input number is an integer, and may be replaced by a larger or smaller scaling factor, depending upon the expected range of inputs. The RST should be replaced by INV Stflg 9 rtn and the R/S at step 000 deleted, if this routine is to be subroutine callable.

Revealed ROM Constants (V3N12p5): John VanWye (982) suggests that the non-normalized constants which Bill Skillman associates with steps 384-447 are not scaled at all (the machine treats them as fixed rather than floating point (V3N7p1,2)). Thus each 8-step register actually holds a 16-place number, with an implied decimal point between the 2 MSDs. But it's not clear how many of the places were intended to be good. John finds ln 1.0001 at steps 424-431 good to al 16 places, but π and π/2 at steps 496-503 and 488-495 respectively appear to be good to only 13 places.

Friendly competition (V3N12p6): Maurice reports receiving a "3" routine from Richard G Snow (brother of Robert G (212)) tying my 11- step routine, but here's a 58/59 one from Jared Weinberger (221) which is 2 steps shorter: INV lnx INV lnx INV lnx y^x lnx x^2 = lnx lnx. (recall that INV lnx counts as only one step).

Bridge deal (V3N10p2): Bill's program has generated at least 2 opposite reactions: Jared considers it insufficiently random at the end, while Lou finds it worthy of replacing his (V3N2p5).

A CROM Identifier: 58/59 users generally working with 2 or more CROMs may find it helpful to insert a CROM-identifying card in the mag/plastic card display slot to remind them which module is in the machine, especially when a CROM-dependent RAM program without a CROM test is run.

Membership Address Changes: 100: C & C of Denver Rm 285 5440 Roslyn St Denver, CO 80216; 372: 572 John Hancock St Orange Park, FL 32073; 606: 2325 N 87th Way Scottsdale, AZ 85251; 751: 500 Newport Ctr Dr #455 Newport Bch, CA 92660; 786: 2416 K St NW #805 Washington DC 20032; 1056: Street address is 7722 (not 722).

An INV HIR Application: Maurice Swinnen (779) notes that although prefixing a HIR operation with INV to change its function may seem to offer no advantage to straight specification, it can save steps in some cases. For example, since INV HIR 31 is the same as HIR 51, and INV HIR 41 is the same as HIR 61, in cases where all 4 register-arithmetic operations are to be subroutine callable, HIR 31 rtn HIR 41 rtn HIR 51 rtn HIR 61 rtn reduces to INV HIR 31 rtn INV HIR 41 rtn, when absolute subroutine addressing either includes or excludes an INV, as desired.

Print Overlays (PC): Roger Cowell (1010) notes that print patterns can be enhanced by repeated transport of the same tape past the printheads. Depending on the length of the tape to be reprinted, some manual re-registration (alignment) may be required.

Volume 4 Number 2 48/39/38 February 1979

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

Transparent CROM Checks (58/59/PC)

Mark Matlock (1077) has found a way to have a CROM-dependent RAM program check for the presence of the required module without being forced to print the module ID (via a CROM check call routine). His approach is to identify absolute subroutine calls unique to each module such that when the correct module is present, the call is transparent (nothing happens), but when any other module is present, the call causes a halt with error condition.

Mark developed a scheme applicable to 5 of the TI CROMs, choosing combinations of Pgm nn and SBR mmm which would be out of bounds for all but one CROM. I've extended Mark's approach to cover the first ten CROMs: 19-588, 22-397, 15-214, 25-246, 30-017, 23-204, 21-441, 12-375 11-587, and 20-306 respectively, where nn-mmm means the sequence Pgm nn SBR mmm. As new CROMs become available, check sequences would need to be revised, a process which can be approached as follows:

1) Construct a table containing the highest step number containing a rtn instruction for each Pgm of each CROM,
2) Starting with the highest numbered Pgm (Marine Navigation (#5) has the most so far) locate the CROM with the largest rtn-step value, and assign this Pgm and SBR to that CROM,
3) Do the same for remaining Pgms, in descending order, skipping those for which the qualifying CROM has already been assigned a check sequence.

This procedure produced all ten sequences for CROMs 1-10 at the point Pgm 11 was reached, but if there were more CROMs, it would have failed to work if Pgm 1 had been reached, and there were outstanding assignments yet to be made. In such a case, judicious "eyeball" assignment exchanges might accommodate a few more CROMs.

Users will need to find a different approach to allow RAM programs using only those CROM routines common to 2 or more CROMs to operate when any one of the required CROMs (but no other) is present. In any case, the more CROMs being considered, the harder it will be to find a complete assignment set.

Custom CROMs present an interesting situation: while there is no reason to suppose that one developer's CROM ID would be guaranteed to be unique among all CROMs (possibly compromising a straight forward CROM check), a custom CROM can be treated like any other with Mark's approach, success being dependent only upon the degree to which for all CROMs concerned, the Pgm tags for long programs are uniformly distributed.

More on CROM RNG Limitations (V4N1p4**)**

Don answered my question, finding an integer seed (in the 0-199017 range) which produces a cycle shorter than m. It turns out that an initial seed of 30073 produces repeating sequences of length 1897 after the first 57 RNs. The seed which generates the 58th RN is 69740.816819 31, and following the 1954th RN the resulting seed is the same real, and the cycle repeats for each successive string of 1897 RNs. So at least one initial integer seed (and probably many more) will produce cycle lengths less than m.

In response to a query by Don, TI suggests that "*In order to achieve a full sequence of 199017 numbers in ML-15, ST-02 and MU-12 ... Step 1 of the user instructions must be replaced as follows ...*", which amounts to writing 26-70 step programs which call selected portions of the CROM RN routines. The net effect is to guarantee that $(ax_n+c) \bmod m$ is an integer before it is used to generate x_{n+1}. This is accomplished by putting Int(r9+½) into Reg 9 before calling SBR D.MS (A for MU-12). (r9 means the contents of Reg 9).

It turns out that you can do about as well or better, memory- and speed-wise by just writing the revised 0-1 RNG into RAM. With a few shortcuts and if truncation to 5 digits isn't required, it might be written: ***Lbl A*** *RCL 10 X RCL 9 + RCL 11 = ÷ RCL 12 = INV Int X x:t RCL 12 + .5 = Int STO 9 x:t rtn,* and run with r10=24298, r11=99991, and r12=199017. But, of course, don't bother with any of this if it doesn't matter how long RN sequence cycles are.

Cycle length is only one of many RN-string characteristics of interest to users. Within a cycle, or fraction thereof, various statistical measures such as mean, standard deviation, runs, and the variously defined distributions along with considerably more arcane mathematical tests can be of interest. Micheal Shunfenthal (1078) has been investigating some of the elementary properties of RN strings produced by the TI routine, and notes that histograms made on RN strings produced by chained seeds show less "randomness" (larger differences between populations in arbitrarily chosen ranges) than for RN strings where each element is generated by a new integer seed. But the seemingly better approach requires generating the integer seeds by some means, and Michael arbitrarily took sequences of the form: n, n-1, ... 0, 2n, 2n-1, ...n, 3n, 3n-1, ...2n, ..., which although not randomly ordered themselves, produce even distributions of RNs by frequency of occurrence. But unfortunately, histograms don't measure run lengths (monotonically increasing or decreasing trends), which for this approach are consistently 8-10 decreasing elements (runs down), making such sequences fail a runs test. With this approach, then, the produced RN string can be guaranteed not to repeat (since each RN is produced by a unique seed), but run trends would tend to be as predictable as is the method for choosing the integer seeds.

But passing a whole battery of tests may not be the best indicator of a good RNG. The best one is most apt to be the shortest and fastest whose RN strings pass those tests critical to a specific application. On the other hand, as Knuth suggests somewhat tongue-in-cheek: "*Perhaps the main reason for doing extensive testing on RNGs is that people misusing Mr X's RNG will hardly ever admit that their programs are at fault: they will blame the RNG, until Mr X can prove to them that his numbers are sufficiently random.*"

Special Case Processing (V3N8p5)

John VanWye (982) has cut Bill's execution time almost in half with the program listed below. John rearranges the original equation to: $(a^3-100a) + (b^3-10b) = -(c^3-c)$, which presents 3 advantages: 1) All the f(c) terms are even, 2) There is no need to set b or c = 8 or 9, and 3) f(c) is never negative. 1) means that only f(a) and f(b) both even or both odd need be summed for comparison with f(c); 2) is confirmed by inspection, and reduces sums and comparisons; and 3) eliminates the need for comparisons when f(a) + f(b) is negative.

In devising a way to synthesize a positional format solution, John found that the units place (c) could be satisfactorily approximated by fix 0 rounding of the cube root of f(c). The tens and hundreds places (b and a) are calculated from the b and a pointers.

TI-58/59/PC Program: Solutions to $a^3+b^3+c^3=100a+10b+c$ VanWye(982)

User Instructions:

Run by pressing A; results in 69 seconds.

Program Listing:

```
000:  RCL*02 + RCL*03 = x:t 336 +/- x≥t 046 210 +/- x≥t 046 120 +/- x≥t 046
      60
030:  +/- x≥t 046 24 +/- x≥t 046 6 +/- x≥t 046 0 x=t 076 2 SUM 02 Dsz 0 000
056:  4 STO 00 8 INV SUM 02 2 SUM 03 Dsz 1 000 Ifflg0 112 Adv R/S +/- y^x 2 1/x
080:  + 10 X (RCL 02 - 4) + 100 X (RCL 03 - 12 = Prt STO 22 0 x=t 146 GTO 049
      4
113:  STO 00 5 STO 01 ST0 03 12 STO 03 Rst Lbl A 4 STO 00 ST0 02 5 STO 01 12
      STO 03 Stflg 0 Fix 0 GTO
144:  000 RCL 22 + 1 = Prt GTO 049
```

Prestored Data:

```
04:  0 -9 -12 -3 24 75 156 273 0 -99 -192 -273 -336 -375 -384 -357 -288
21:  -171
```

Although no one has yet responded to the generalized problem suggested in V3N7p3, Gunter Merten (750) has looked at problems of the form: $a^3+b^3+c^3= 10^4a+10^2b+c$ where a is in the 10-99 range and b and c in the 0-99 range. Gunter also extends this to $a^3+b^3+c^3= 10^6a+10^3b+c$ and $= 10^8a+ 10^4b+c$, with a,b,c range limits increased each time by a factor of ten. He offers sample solutions of $16^3+50^3+33^3= 165033$, $166^3+500^3+333^3= 166500333$, and $1666^3+5000^3+3333^3=166650003333$,and challenges PPC users to devise efficient approaches to finding all the solutions.

Efficient Data Packing (V2N11p5)

Arthur Ehrlich (969) poses a requirement to pack and unpack up to 8 integers in the 2-24 range per register. This is the maximum possible, using the V2N11p6 formula: Int(12 ÷ log25), but in order to provide for random/multiple stores and recalls, a more elaborate approach than outlined in V2N11 would need to be found.

The Math/Utilities MU-08 program might be used as a start. What it lacks is the means to change the radix of the pack/unpack arithmetic. Members are invited to try revising MU-08 (or to try another approach which produces as general-purpose a routine) so that n-digit numbers whose maximum values are less than the base ten maximum can be more efficiently packed. I doubt that it will be easy: MU-08 does a lot of data manipulation to allow random and multiple stores, recalls and exchanges. Associated with each input datum is a key (V3N2p3,4) which TI refers to as a pseudo register (PR) number, which may be any of 1,2, ... n where max n is determined by field sizes and the number of available full data registers. To specify the packing format, key a.bc..., press A, where a is the number of data to be packed per register, and b,c, ...

each specify a field width in the 1-9 range for each of the a data. The total of b+c+ ... must be less than 14. For example, a 3.246 format specifies 3 data per register, the first up to 2 digits wide, the second up to 4 digits, and the third up to 6, adding up to 12 (one less than the max). To store a datum, key it (a positive integer), press x:t, key the PR number you wish to be its "key", press B; to recall, key the PR number, press C; to exchange, key the new datum, press x:t, key the PR, press D, and see the old one displayed. For a variable radix version, I expect practical considerations would require all data fields to be the same length. Anyway, here is a listing of MU-08:

```
000:  Lbl A STO 01 rtn ((CE - 1) ÷ RCL 01 STO 00 Int INV SUM 00) STO 02 (INV
      Int X RCL 01 Int)
031:  STO 03 Op 23 4 SUM 02 RCL*02 INV Int STO*02 (INV Dsz 3 067 (RCL 00 X
      10) STO 00 Int
060:  INV SUM 00 + GTO 045 0) INV Log D.MS STO 03 Prd*02 rtn Lbl B SBR 005
      RCL*02 Int
085:  INV SUM*02 (Exc 00 X 10) Int INV Log DMS Prd 00 Prd*02 Prd 03 ((1/x X
      x:t
110:  ) (INV Int ÷ x:t) Int + Exc0 + RCL*2 INV Int) STO*2 RCL3 INV Prd*2 R0
136:  rtn Lbl C SBR 005 (RCL*02 INV Int X RCL 03 INV Prd*02 (RCL00 X 10) Int
      INV Log DMS)
165:  Int rtn Lbl D SBR 005 RCL*02 Int INV SUM*02 (Exc 00 X 10) Int INV Log
      DMS Prd 00
191:  Prd 03 Prd*02 (1/x X x:t) (INV Int ÷ x:t) ((Int + RCL*02 INV Int + RCL
      00) ÷
219:  RCL 03) Exc*2 Int rtn
```

Editorial: Looking Ahead

Since I'm currently about out of those member-inputs which I consider worthy of publication, this will be the last regular monthly 52-NOTES (unless a lot of good material starts arriving soon: useful discoveries and inventions, clever routines, and programs of broad interest which demonstrate new (better) programming techniques). In the past, there have been occasions when I've almost set aside real gems because descriptive material was poorly expressed or nonexistent, and I expect some goodies have been languishing unpublished because their value escaped me on first glance. So if you've sent me something you feel is likely to meet my criteria, but which hasn't yet seen print, let me know. But please make an effort to establish its originality (scan back issues of 52-NOTES) and identify the important new features.

It may be that after almost 2 years, we've just about covered the newer PPCs (there didn't appear to be much more to be said about the 52 or 56 after they were 2 years old, or so), and so far no news of any 58/59 or 57 successors has come to my attention. While the long-awaited TI personal microcomputer may make its debut some time this year, there are indications of more schedule slippages. In the meantime, it will help me to decide whether to broaden 52-NOTES coverage to include micros, if each of you will convey your interest: pro or con, and in which machines. Even though there is currently a lot of micro coverage in a growing proliferation of periodicals, perhaps there is an unfilled place left for 52-NOTES' style and technical level.

In any event, it is my intention to continue publication, but on an irregular schedule if need be, determined by and large by the rate at which I receive good inputs. At such time as the scope settles down, I'll consider Club and newsletter name changes. Any member who wishes to terminate his membership now may send me a SASE (less stamps for members abroad) for a refund of outstanding contributions. Those wishing to continue should consider their memberships linked to the number of issues received after V4N2. I suggest that you record the number of issues due you now, and to Dsz it each time you receive a new issue. Program the zero-skip to remind you when to contribute again! The original contribution rates continue to be adequate, and back issues will be made available at the same rates, for the forseeable future.

Let me close by saying that I continue to enjoy running the Club and editing and publishing 52-NOTES, and hope that inputs will increase to the extent needed to continue (or get back on) a regular monthly basis.

Interpolation and Extrapolation

There are many occasions in science and engineering when for a given set of data points there are requirements to interpolate (find additional points in between the given ones) and/or extrapolate (find points outside the span of the given set). In both cases values for the added points are generally determined by one of two means: 1) A curve is generated which passes exactly through all the given points, or 2) A "best-fit" curve is generated to pass close to the given points. In cases where there is reason to believe that all given points are "correct" (very accurate), it may be best to force an exact fit through them, and this can be done for n points by a polynomial of degree n-1. On the other hand, in cases where there is significant noise in the data and/or many points, the second way is apt to be better, and the method of least squares is commonly used.

Bill Skillman (710) has written a short fast Polynomial Least Squares Fit program to which I added a transparent module test (V4N2p1), and which fits an nth degree polynomial to n+1 or more data points for n in the 1-6 range. It runs on a 59, either with or without the PC, but with it, without tags, to hold program length down to one card-side. Members able to squeeze in tags without overflowing to another card-side are invited to share their approaches.

TI-59(PC) Program: Polynomial Least Squares Fit B.Skillman(710)

User instructions:

Key order n (less than 7), display flashes if ML module is not connected; press A; key x,y pairs: x_i, press B, y_i, press R/S, repeat for at least n+1 pairs. Press C, see a0; press R/S, see ai, repeat for i=1,2, ...n. With printer, order and inputs are confirmed, followed by an unsuppressable determinant, and then the coefficients a_0, a_1, ...a_n. To interpolate or extrapolate, key x, press E, see y displayed and/or printed.

Program Listing:

```
000:  Lbl E Pgm 7 C rtn Lbl A Pgm 19 SBR 588 CMs STO 07 Prt X (CE + 5) + 11 =
      ST0 00
029:  9 Op 17 1 STO 08 rtn Lbl B STO 02 Prt RCL 00 ST0 04 9 STO 05 R/S STO 03
      Prt SUM*04 RCL 07 STO 06 ST0 01
060:  1 X RCL 02 X SUM*5 x:t Op 25 RCL 03 = Op 34 SUM*04 x:t Dsz 6 061 X RCL
      02 = SUM*05
088:  Op 25 Dsz 1 082 Op 28 RCL 08 rtn Lbl C Op 38 RCL07 + 8 + STO 01 RCL 07
      STO 04 STO 02 x^2 =
118:  SUM 02 RCL 07 SUM1 STO3 Op23 RCL*01 STO*02 Op 31 Op 32 Dsz 3 128 Dsz 4
      120 Op 27 Pgm 02
148:  C RCL 00 STO 04 RCL 07 STO 02 1 Pgm 02 D RCL 07 x^2 + 7 = STO 01 RCL 05
      x:t Op 21 RCL*01 INV x=t
178:  172 RCL 07 SUM 01 Op 25 0 Exc*04 STO*01 Op 34 Dsz 2 161 Pgm 02 E 1 Pgm
      02 A' RCL 07 STO 00
208:  STO 04 Op 34 5 STO 02 Pgm 02 R/S STO*02 SBR 236 Op 22 Dsz 0 215 6 Op 17
      Adv Adv Adv rtn Op 08 R/S rtn
```

The program which follows, fits an nth degree polynomial to exactly n+1 data points, following an algorithm and parts of a FORTRAN implementation given in **Data Reduction And Error Analysis For The Physical Sciences** by Philip R Bevington; McGraw-Hill, 1969 pp264 and 267. Incidently, chapter 8 of this book describes the least squares fit method.

TI-59(PC) Program: Variable Spacing Interp. and Extrap. Ed

User Instructions: Key x1, press x:t, key y1, press E; key xi, press x:t, key yi, press R/S, repeat for i=2,3, ... LT 11. Initiate processing: press A. To interpolate/extrapolate, key x, press C. With printer inputs are tag-confirmed, and outputs tagged; either with or without printer, inputs are followed by i displayed; processing ends with zero displayed, and each interpolated/extrapolated y is displayed following C processing.

Program Listing:

```
000: Lbl E CMs STO 21 STO 50 x:t STO 11 12 STO 00 22 STO 01 1 ST0 02 44 Op 04
     RCL 11 Op 06 45 Op 04
033: RCL 21 Op 06 Lbl 1' R/S STO*01 x:t STO*00 44 Op 04 RCL*00 Op 06 45 Op 04
     RCL*01 Op 06 Op 20
063: Op 21 Op 22 RCL 02 GTO 1' Lbl A 31 STO 04 RCL 02 STO 05 RCL 12 – RCL 11
     = STO 03 11 STO 00 Lbl 2' RCL*00
097: - RCL 11 = ÷ RCL 3 = STO*4 Op 24 Op 20 Dsz 5 2' RCL 02 - 1 = ST0 05 2
     STO 08 RCL 21 STO 41 Lbl 3'
130: 1 STO 52 0 STO 51 RCL 08 - 1 STO 07 = STO 09 Lbl 4' RCL 08 – RCL 04 =
     STO 06 30 + RCL 08 = STO 04 RCL*04 -
166: (30 + RCL 06) STO 04 RCL*04 = Prd 52 40 + RCL 06 = STO 53 RCL*53 ÷ RCL
     52 = INV SUM 51 Op 27
199: Dsz 9 4' 20 + RCL 8 = STO 53 RCL*53 ÷ RCL 52 + RCL 51 + (40 + RCL 8) STO
     53 0 = STO*53
232: Op 28 Dsz 5 3' CLR Adv R/S Lbl C x:t 67 Op 04 x:t Op 06 – RCL 11 = ÷ RCL
     03 = STO 54
260: RCL 41 STO 51 2 STO 06 RCL 02 - 1 = STO 05 Lbl 5' 1 STO 52 RCL 06 - 1 =
     STO 09 1 STO 07 Lbl 6' RCL 54 – (30
297: + RCL 07) STO 04 RCL*4 = Prd 52 Op 27 Dsz 9 6' 40 + RCL 06 = STO 53
     RCL*53 X RCL 52 = SUM 51
329: Op 26 Dsz 5 5' 45 Op 04 RCL 51 Op6 Adv Adv Adv R/S
```

Record with turn-on partition in Banks 1 and 2.

Tips and Miscellany

Off-The-Shelf Custom CROMs (V4N1p1): Bill Fagerstrom (692) notes that Datalab, Inc Box 292 Haverford, PA 19041 is marketing a custom CROM for securities traders called the Options Analyst Datalab Mod-1 Library module. The module, keyboard overlay, users manual, and a 6 months newsletter subscription sell as a minimum package for $225.

Euclid's Algorithm Routine (V3N11p5): Carl Seel (328) notes that the EE INV EE rounding doesn't properly process some (relatively prime) inputs. In such cases the mantissa is too large for the EE (with turn-on fix) to round, and Carl suggests substituting fix 0 D.MS INV fix. However, D.MS is noticeably slower than EE, and it turns out that the V3N11p5 routine as written appears to work for all cases is run with a fix 0 display.

Forced Card-Read(59): John Allen (104) notes that contrary to a statement on page VII-5 of the users manual, following a forced card-read, the display remains unchanged.

Friendly competition (V4N1p6): Philip Morey (1129) worked out Jared's 9-step solution independently, and shows that HP machines can also produce a 3 in 9 steps: e^x e^x *ENT* x^2 e^x *x:y* y^x *Ln Ln*.

Membership Address Changes: 954: 311A San Francisco Blvd San Anselmo, CA 94960; 959: 620 Iris Ave #410 Sunnyvale, CA 94086; 1003: 529 4th Ave Bethlehem, PA 18018; 1056: 315 Tumblebrook Slidell, LA 70458.

Correction (V3N12p2): Jared Weinberger (221) notes that there is an INV missing at step 707 (between Op 8 and Ifflg 3).

Print Borders(58/59/PC): Richard Snow (212B) suggests using the exchange symbol (print code 62) in the construction of vertical lines. This, in conjunction with the dash (code 20) for horizontal lines, and the + (code 47) for corners or intersections makes an attractive rectangular border, or tic tac toe grid.

```
*****    ***          *     *   ****   *****  *****   ***
*       *   *         **    *   *   *    *    *      *   *
*           *         * *   *   *   *    *    *      *
****       *    ****  *  *  *   *   *    *    ***     ***
    *     *           *   * *  *   *    *    *         *
    *    *            *    **  *   *    *    *      *   *
****    *****         *     *   ****    *    ******  ***
```

Volume 4 Number 3 48/39/38 March-May 1979

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

A Keyboard SBR N in One Step (58/59)

Ed Westenhaver (1095) has discovered another useful end-of-partition function (V3N6p5), noting that a program-executed SBR at the last step presents the user with the ability to specify continued execution at any defined label without manually having to key SBR. For example, with a 59 at turn-on, and the following sequence written in RAM: *Lbl A GTO 479... Lbl RCL... R/S... 479: SBR*, pressing A causes a halt with flashing display. If RCL is then pressed, the code following Lbl RCL is executed.

Mechanizing this discovery in programs with many entry points can save a significant number of input keystrokes. Ed found an improved typewrites (V2N9p5) to be a good example. The program which follows is based on one Ed wrote, combining his key-character assignments with faster processing. The space, and alphabet characters can be input at a rate of about one per second (compared with about one per 2 seconds for the TI LE and MU CROM routines), and require only a single keystroke per letter. The remaining 37 characters require the shift key (2nd) as a prefix, and can take as long as 2 seconds for input processing, depending upon how far down in RAM their labels are.

I've found that the plastic key-code overlay (factory supplied with each 58 or 59) makes a good overlay base on which adhesive paper strips can be mounted, and appropriately inscribed as follows:

A =start/Π		C =delete	D =next	E =sp/II
2nd =2nd	INV =A/Σ	lnx =B/Δ	CE =C/x	CLR =D/x
	x:t =E/↔	x^2 =F/^{2}	√x =G/√	1/x =H/↑
	STO =I/,	RCL = J/!	SUM =K/*	y^x =L/^{x}
	EE =M/%	(=N/(	) =O/)	÷ =P/÷
GTO =Z//	7 =/7	8 =/8	9 =/9	x =Q/x
SBR =Y/'	4 =/4	5 =/5	6 =/6	- =R/-
RST =X/?	1 =/1	2 =/2	3 =/3	+ =S/+
R/S =W/e	0 =/0	. =V/.	+/- =U/π	= =T/=

where for a=b/c, a is the stamped name of a key, b is the assigned first function and c the assigned second function. The B key is free to be used for either first or second function assignment to one or two frequently used characters, in which case B and/or B' could replace the currently assigned labels for those characters.

Members may wish to modify this program to store assembled printcode for data recording, and are invited to try to speed up execution.

TI-59/PC Program: Fast Typewriter Ed Westenhaver (1095)/Ed

User Instructions:

```
Press A to initialize; see number of character-positions remaining (20)
```

flashed. Input characters per overlay. Full lines are automatically printed; press D to initiate the printing of less than 20 characters. Press C to backspace, and press LRN LRN to clear the flashing error-state.

Program Listing:

```
000: STO*00 Op 20 Dsz1 477 GTO 323 Lbl E 0 RST Lbl INV 13 RST Lbl lnx 14 RST
     Lbl CE
027: 15 RST Lbl CLR 16 RST Lbl x:t 17 RST Lbl x² 21 RST Lbl √x 22 RST Lbl 1/x
     23
054: RST Lbl STO 24 RST Lbl RCL 25 RST Lbl SUM 26 RST Lbl yˣ 27 RST L EE 30
     RST
080: Lbl ( 31 RST Lbl ) 32 RST Lbl ÷ 33 RST Lbl X 34 RST Lbl - 35 RST Lbl +
     36 RST
110: Lbl = 37 RST Lbl +/- 41 RST Lbl . 42 RST Lbl R/S 43 RST Lbl RST 44 RST
     Lbl SBR
137: 45 RST Lbl GTO 46 RST Lbl Dsz 1 RST Lbl Ifflg 2 RST Lbl DMS 3 RST Lbl π
     4 RST
161: Lbl x≥t 5 RST Lbl Σ+ 6 RST Lbl x 7 RST Lbl x=t 10 RST Lbl Nop 11 RST Lbl
     Op
187: RST Lbl Adv 40 RST Lbl Prt 53 RST Lbl List 64 RST Lbl Stflg 71 RST Lbl
     Grad
210: 47 RST Lbl Lbl 65 RST Lbl Rad 20 RST Lbl Pause 63 RST Lbl Deg 50 RST Lbl
     Eng
235: 61 RST Lbl Fix 55 RST Lbl Int 56 RST Lbl Abs 72 RST Lbl CMs 57 RST Lbl
     Exc 73
262: RST Lbl Prd 51 RST Lbl Cos 52 RST Lbl Write 54 RST Lbl Sin 70 RST Lbl
     P/R 62
287: RST Lbl Tan 60 RST Lbl CLR' 66 RST Lbl CP 67 RST Lbl log 75 RST Lbl INV'
     77
312: RST Lbl E' 74 RST Lbl A' 76 RST Lbl D Op 00 RCL 06 + RCL 05 EE 2 INV EE
     + RCL 04 EE 4 INV
342: EE + RCL 03 EE 6 INV EE + RCL 02 EE 8 = INV EE Op 01 RCL 11 + RCL 10 EE
     2 INV EE +
370: RCL 09 EE 4 INV EE + RCL 08 EE 6 INV EE + RCL 07 EE 8 = INV EE Op 02 RCL
     16 + RCL 15
398: EE 2 INV EE + RCL 14 EE 4 INV EE + RCL 13 EE 6 INV EE + RCL 12 EE 8 =
     INV EE
424: Op 03 RCL 21 + RCL 20 EE 2 INV EE + RCL 19 EE 4 INV EE + RCL 18 EE 6 INV
     EE + RCL 17
452: EE 8 = INV EE Op 04 Op 05 Lbl A CMs 2 STO 00 19 STO 01 Lbl C Op 30 Op 21
     RCL 01 SBR
```

Vectored Processing with Op 10 (V1N3p5, V1N4p3, V2N8p1, V2N11p2, V3N2p3, and throughout the calendar printing articles: V3N5-V3N9)

Past articles have discussed various methods by which the input datum is used either directly or following a transformation to "point" to one of several processing choices, in addition to its possible function as a processing input. The primary advantage of such mechanizations is fast processing: time is not wasted conducting multiple comparisons. For example, consider a program designed to accept one of ten possible inputs in the 0-10 range, where for an input 1, processing is to begin at step 10, for 2 at step 20,... for 10 at step 100. One way to accomplish this is to compare the input successively with 1,2,... 9 and to branch to the desired address when a match is found. But this approach is slow and code-consuming compared with a method which branches indirectly on ten times the input as a pointer. The first approach might look like: ***Lbl A*** *x:t 1 x=t 010... 9 x=t 090...,* and the second like: ***Lbl A*** *STO 0 X 10 = STO 1 GTO*1...* . Here, the input is vectored to the desired processing via a simple transformation (ten times the input).

Other more complicated transformations have been discussed in earlier articles, involving carefully contrived arithmetic manipulations to eliminate, or at least minimize "collisions" (V2N8 and **Scientific American** Apr 77 p63-80). For the special case

where there are only 3 processing choices to be made, Jared Weinberger (221) and Bill Skillman (710) have been exploring a new approach using the 58/59 Op 10 function.

From: ***LBL A*** *(CE + Op10 STO 4 Op24 RC*4 STO 04 0) GTO*4* by Jared which handles a 3-way branch on a negative, zero, or positive input through Reg 0, 1, and 2 as pointers, Bill has devised eleven related routines to cover all possible conditional relationships on both a single variable WRT zero, and 2 variables WRT each other:

```
Lbl A (CE + Op10 STO4 Op24 RC*4 STO4 0) GTO*4 Lbl B (CE + Op10 STO4 RCL*4
STO4 0) GTO*4 Lbl C (CE + Op10 +/- STO4 RCL*4 STO4 0) GTO*4 Lbl D (CE + Op10
Abs STO4 RCL*4 STO4 0) GTO*4 Lbl E (CE – RCL 3 + Op10 STO4 Op24 RCL*4 STO4
RCL3) GTO*4 Lbl A' (CE – RCL3 + Op10 STO4 RCL*4 STO4 RCL3) GTO*4 Lbl B' (CE –
RCL3 + Op10 +/- STO4 RCL*4 STO4 RCL3) GTO*4 Lbl C' (CE – RCL3 + Op10 Abs STO4
RCL*4 STO 4 RCL 3) GTO*4 Lbl D' (RCL5 – RCL3) Op10 STO4 Op 24 RCL*4 STO4
GTO*4 Lbl E' (RCL5 - RCL3) Op 10 STO4 RCL*4 GTO*4 Lbl + (RCL5 - RCL3) Op10
+/- STO4 RCL*4 STO4 GTO*4 Lbl = (RCL5 - RCL3) Op10 Abs STO4 RCL*4 STO4 GTO*4.
```

In all cases prestore 2-way branch addresses in Reg 0, 1, and 3-way branches in Reg 0, 1, and 2.

It would appear that practical applications of this approach would be in cases where no suitable hash function can be found, and the T-register is needed for other purposes. Members are invited to make improvements and/or report their own practical applications.

In another application of Op 10, Jared beats the TI 58/59 Manual's IV-96 check-service-charge program with: ***LBL A*** *STO0 fix2 X .1 A' A'* ***Lbl A'*** *– 5 INV SUM0 RCL0 X (Op10 + 1) ÷ 200 = rtn* which is shorter, and faster for inputs greater than 14.

Book Review: Countdown: Skydiver, Rocket and Satellite Motion on Programmable Calculators by Robert Eisberg and Wendell Hyde (Dilithium Press Box 92 Forest Grove, OR 97116; 1979, 107pp $6.95)

This new publication appears to be a sequel to Eisberg's earlier work: Applied Mathematical Physics With Pocket Programmable Calculators (V2N1p6), simplifying the sections on freefall, fall with friction, and central force motion, targeting the algorithms to the newer low-end PPCs: TI-57 and HP-33E.

The reader is assumed to be new to both physics and programming and is led step by step in both areas to mechanize a few simple numerical solutions to some of the fundamental motion problems. The chosen applications, and variations on them should be both entertaining and educational to the physics/programming novice, and without burdening him with the formal definitions, the text announces at the end that the reader/user has solved nonlinear second order, and "... coupled, second order, differential equations. Congratulations."

More on the MU CROM (V3N12p3-5)

Bill Skillman (710) has examined and used the routines in this CROM, and offers the following comments:

MU-04: Columns of figures can be printed without using MU-03. (I note also that with or without MU-03, restarting using old line numbers required CMs to "erase" previous characters).

MU-06: If data are prestored in Reg 1 to n by non-MU-06 means, Reg 0 needs to be initialized with n.

MU-08: The max number of pseudos per real register may be extended to 11 by creating the format statement incrementally (register arithmetic, or display summing (see V3N7p4)).

MU-11: There is noticeable accuracy degradation for inputs less than ½.

MU-14: For call by a user program, follow manual initialization; the suggested shortcut in incomplete.

MU-15: A factor of x was omitted in the denominator of the f'(x) expression, $x \neq 0$.

MU-17: Contrary to instructions, you can use = and CLR in SBR A' but you must preserve t (MU-17 uses T in an x≥t comparison at step 236)

Two observations which Bill makes generalize to all CROMs (as noted by Fred Fish (V3N3p1)): 1) Check CROM listings for routines which only store inputs, and for user program calls, bypass those with appropriate user sequences of the form: STO nn; 2) Check CROM code before and after calls to user routines (via Pgm 00 N) to see if it is safe to use CLR and/or =, since it is not always necessary to heed related caveats in the CROM manuals.

The MU CROM is slowly becoming available (not yet showing up in Dayton), and as more members acquire it, I hope to get more discussion of this unusually good module.

***** David W Johnston (5) -1979 *****

I was saddened to learn via a note from Jim Doman (473) of Dave's death 2 April 1979. In his last letter to me in January, Dave reported that he was undergoing chemotherapy for cancer, and that he was "holding my own".

Dave will be missed, especially by those members who have been active in the 56/57/58 Program Exchange which he ran so efficiently and modestly. I have been in touch with Dave's closest survivor, a cousin living in Massachusetts, and hope to obtain Dave's Program Exchange records. At such time as they become available, I'll pass them along to a volunteer, or lacking the latter will attempt to carry on the Program Exchange Service myself.

Another PPC notable died in January: Jim Davidson, a major contributor of input to **65-Notes/PPC Journal**, and a participant in our Friendly Competition (V2N12p4).

Zero as a Callable Label (58/59)

Maurice Swinnen (779) passes along a discovery published in **Display** (V5N5+6S76), which reveals that zero can be used as an addressable label (V3N7p3). It turns out that a call to any of the ten user-defined keys (A-E') preceded by Ind, executes as a call to Label 0 (if defined). The user-defined label may (but need not) be defined. For example: ***Lbl A*** *seq1 rtn* ***Lbl 0*** *seq2 rtn* called by pressing A, executes seq1, while Ind A (or Ind B,C,...E') does seq2.

A call to ***Lbl A*** *seq1 Ind C seq3 rtn* ***Lbl 0*** *seq 2 rtn* with A does seq1 seq2 seq3. Zero appears to be the only numeral which will work this way, and somehow Reg 0 gets into the act, since a print trace reveals its contents following execution of Ind n (n=A,B,...E') as if Reg 0 were being addressed indirectly.

ROM Constants Update (V4N1p6)

John has explored further and reports: "I have determined the accuracy of all 16 of them. Nine turned out to be good to all 16 places; the other seven are good to the number of places shown ( ): $\tan^{-1}.1$ (14), $\tan^{-1}.01$ (15), ln 10 (14), π (13), π/2 (14), π/4 (15), and 18/π (13).

"The ROM constants for $\tan^{-1}.1$, $\tan^{-1}.01$, and π/4 end in zero, and appear to be simply a rounding to the number of digits given. The other four all end either 012 (ln 10, π), or 022 (π/2, 18/π), so these numbers could be some sort of code. (that the 14th digit on ln 10 (0) is good is probably a coincidence)."

Corrections: V4N2p5 step 234 is Adv; V4N2p6 step 254 is ÷ (divide) and step 263 is 51."

Membership Address Changes: 396: 569 Broadway Ave Victoria BC V8Z 2G3 Canada; 559: 4427 Rosada St Long Beach CA 90815

52-Notes - Table of Contents

    - Zero Divide Zero
- Bagels One to Ten Anomaly (see program in V1N1p6)
- Notations/Conventions
- Membership
- Machine Differences
- HP-67

Volume 1 Number 3 -- August 1976

- PC-100 Printer Techniques
- Pointer Rules
- SR-52 Program: Printer Graphics
- Letter Game Conventions
- The NIM Games
    - SR-52 Program: Dynamic NIM
- Dix Fulton'S 4 X 4 Determinant Program
    - SR-52 Program: 4 X 4 Determinant
- Automatic Card Read
- Advanced Programming Techniques (Part I: ICS)
- Lack of Reg 60-69 Access
- SR-52 Material in 65-Notes
- PC-100 Hardware Tips
- Dim Display
- Integer/Fraction Truncation
- TI's Computer Monitored Repair Service

Volume 1 Number 4 -- September 1976

- TI Announces its Professional Program Exchange (PPX) for the SR-52
- New SR-52 Owners Manual
- Editorial
- More On LBL LBL
- Number Range Notation
- Table Lookup Applications
    - SR-52 Program: Solution To 4 Simultaneous Equations
- A Shooting Stars Game (Using Vectored Processing)
    - SR-52 Program: Shooting Stars (for PC-100 printer)
- Diagnostic Programs
- Programmed Card Write
- Tips
    - Efficient Handling of Constants
    - Physical Effects of the Printer on the SR-52
    - Mag Card Care
- Efficient Coordinate Conversion Processing
    - SR-52 Program: UT/AZ/ALT/Range from NASA Bulletins
- Codes for Mantissa and Exponent Signs
- Streamlined Dynamic NIM
    - SR-52 Program: Dynamic NIM (revised)
- Routines
    - Fractured Display under Program Control
    - Exponent Extractor
- Calculate Mode Flag Test
- Membership Address Changes/Corrections

Volume 1 Number 5 -- October 1976

- Editorial
- TI's SR-52 Programming Workbook
- PPX-52 Status
- Shooting Stars Solutions
- Shooting Stars Program Addendum
- SR-52 "Crash"
- Pseudos
- Pseudos (con)
- Advanced Programming Techniques (Part II: Table Lookup Optimization)
- Table Lookup Optimization (con)
- Tips
    - Battery Charger Connecting
    - Exchanging Mag Cards
    - Mag Card Static Charge
- Routines
    - Fixed Point Extractor

Volume 1 Number 6 -- November 1976

- Decapower
- Forum
    - Programming Help
    - Implied Multiplication
- Machine Interface Specifications
- Tips
    - Manual Interaction with an Executing Program
    - The Use of Inexact Functions To Produce Integers
    - Short, non-Pseudo Error-State Producers
    - Use of Flags 5-9
    - An Executable Separation of Register Operator and Operand
    - A New Facet to the CLR Function
- Routines
    - Displaying 11th, 12th and 13th Digits
    - Timed "Crash"
- A Difference-Table Program Using the PC-100 Printer
    - SR-52 Program: Difference Tables (with PC-100 Printer)
- The "Revised" SR-52 Owner's Manual
- Local Clubs
- Register Exchange for Data Recording and Retrieval
    - SR-52 Program: Register Exchange
- SR-56 Program Exchange
- Table Lookup Optimization Addendum
- More on Displaying 11th 12th and 13th Digits
- More on Shooting Stars (V1N4p3)

Volume 1 Number 7 -- December 1976

- Season's Greetings to All!
- "Rounding" Terminology
- The First TI PPX-52 Catalog
- Bringing the SR-56 into SR-52 Discussions
- Forum (52)
- Tips

        - Use of Pseudo 73 (52)
        - Short Absolute Branch (52)
        - Label Execution Anomaly (52)
        - Electro/Mechanical Tips
        - Writing Good Diagnostic Programs (52)
        - Execution Time Variations with Formatting
        - Revision to Automatic Fill of Reg 60-69 (V1N1p3) (52)
        - More on CLR (52)
        - More on Timed Crash (52)
        - More on INV Viability (52)
        - Charger Connection
        - More on the 0 div 0 Error State
    - Membership Address Changes/Corrections
    - One More (The Last?!) on Shooting Stars
    - Routines
        - Pending Parenthesis Extractor
        - A Partial Wipeout (52)
        - *INV' Crash
        - More on Last Digits Viewers (52)
    - Valid Comparison of Two Numbers
    - A Clever D/R Switch Interrupt Processing Application
        - SR-52 Program: SR-52 Timer

Volume 2 Number 1 -- January 1977

- Puzzling Sequences
    - A *LBL Peculiarity (52)
    - *INV' Acting Like p21
    - SST'd SST (52)
    - .001 Dim Display
- Alphabetized Membership List
- TI Notes
    - Machine Anomalies
    - Games PAC
    - PPX-52 Publications
    - Machine Exchange
- Corrections
    - SR-52 Timer Instructions (V1N7p6)
    - Routine B (V1N5p5)
- Register Behavior: Part III Overflow And Underflow
- The Matrix Challenge (52)
    - SR-52 Program: 4 Simultaneous Equations
    - SR-52 Program: 4 X 4 Determinant and Inverse
- Forum
    - PC-100 Hardware Problems
    - Duplicate PPX-52 Programs
    - SR-52 Pause Function
- Book Review: Applied Mathematical Physics with Programmable Pocket Calculators
- Correction: V2N1p4 (bottom)
- Membership Address Change

Volume 2 Number 2 -- February 1977

- Tips

        - Clever Uses of INV P/R
        - More on Charger Connection
        - A Decrement-Only dsz (52)
        - 1/0=0
        - Trig Function Anomalies
        - More on + STO 6O (52)
        - The Printed "?" as an Identifier (PC-100)
        - Step 223 (99) *list
        - Optimized Hardware Interrupts
        - Re-softening a Hardened Display
    - Routines
        - Short Fibonacci Sequence (56)
        - Short, Fast Quadratic Solutions
    - Analysis of a Super Program
        - SR-52 Program: 5 X 5 Determinant and Inverse; 4 x 4 Determinant and (with PC-100) 4 Simultaneous Equations
    - Magazine Review
        - Byte
        - Popular Computing
    - A Fractured Digits, All Flags, Game Application (52)
        - SR-52 Program: Yahtzee
    - Forum
        - Sick Machine Exchange
        - Applications Topics
    - SR-56 Program Exchange Update (See V1N6p6)
    - Miscellany
        - Corrections to TI Software
        - Display Arithmetic Modification
        - SR-52 Successor
        - Program Listing Conventions
        - A New Facet to Registers 20-59
        - SR-60 Notes
        - Mode Terminology
        - Membership Address Changes

Volume 2 Number 3 -- March 1977

- Random Numbers
- A Sum of the Digits Challenge
- PPX-52 Gems and Rejects (52)
- Book Review
- A Subtle Danger in Using "iferr" as a Flag (52)
- A Utility Program for Fractured Digits (52)
    - SR-52 Utility Program: Display Variations Synthesizer
- An AOS Advantage for a Problem in Logic (52)
    - SR-52 Program: Truth Tables
- TI Notes
    - SR-52 and SR-52A Trouble Shooting Guide
    - PC-100 Modifications
- Routines
    - An Efficient Input Routine (52)
    - A Short Integer Test

Volume 2 Number 4 -- April 1977

- Editorial: A Perspective On Machine Comparisons and Loyalties
- More on Printer Graphics (52)
- Membership Renewals
- A Game Challenge
    - SR-52 Program: Corner The Lady Game (Fibonacci Notation Version)
    - SR-56 Program: Corner The Lady Game (Standard Chessboard Version)
- Book Review: SR-52 and SR-52A Troubleshooting Guide
- Correction
- Pseudo 83: Part I: Accessing the Instruction Address Register (lAR)
- SR-56 Program Exchange
- TI Invitation to Club Members

Volume 2 Number 5 -- May 1977

- Quasi-Alphanumerics: A Comparison Of SR-52 And HP-67 Capabilities
    - SR-52 Program: SR-52 Message Synthesizer
- SR-52A/PC-100A
- TI Notes
    - Machine Exchange at TI Centers
    - Adding Chips (52)
- Tips
    - Mag Card Marking
    - Another One-Step Error Producer
    - More on Trig Function Anomalies
    - IND Prefix to a Subroutine Call (52)
    - Battery Charging PAC
    - More on the 0 Divide Error Condition (52)
    - Fast Relative Addressing (52)
- Routines
    - Odd/Even Determiner (52)
    - Efficient Temperature Units Conversions (52)
- Membership Address Changes
- Friendly Competition
    - SR-52/PC-100 Program: 5 X 5 Determinant and Inverse
    - SR-52 Program: 70 Digit Square
- PPX-52 Gems
    - #300032A: 4 X 4 Matrix Multiplication
    - #910050A: Two to Nine Pile NIM
- Book Review: Computers At Large
- Meeting the Sum-of-the-Digits Challenge (V2N3p2)
- Subject/Author Cross Reference

Volume 2 Number 6 -- June 1977

- The New TI PPCs
- TI-59 Program: Interactive Arithmetic Teacher
- Pseudo Behavior Summary (52)
- Hardware Modifications
- More on Corner the Lady (Wythoff's NIM) (52)
- A Few Tips on Presenting Your Inputs to 52-Notes
- SR-52 Program Review: Analytic Computer Model
- Tips
    - A y^x Workaround

    - A Decrement Only dsz (56)
    - 0 divide Error State (56)
- TI Notes
- Coping with Built-in Function Anomalies

Volume 2 Number 7 -- July 1977

- A Few Observations on the Occasion of our First Anniversary
    - TI-58/59/PC-100A Program: Enhanced ML-02 Program
- 0^x Definitions
- Hyperbolic Functions Shortcut
- More on Reg 60, 61, ...69 Behavior (52)
- Pseudo 73 Limitations (52)
- Step 223 Odd Behavior (52)
- Information Referrals
- More on Don Ellis' Publications (V2N3p3)
- More on Sum-of-the-Digits (V2N5p6)
- Decapower Zero Suppression Following Negative Tests (52)
- SR-56 Program Exchange Update (V1N6p6)
- Machine Coverage in 52-Notes
- Data Entry Sensing
- Membership List Corrections
- Assessment of the 58/59 Master Library CROM Programs
- 58/59 Tips
    - Short Form Addressing
    - ML-01 Print Routine Use
    - Taking Advantage of Control Operations
    - INV and Ind Combinations
    - The Nop Instruction
    - Getting Merged Code into the Last Partitioned Step

Volume 2 Number 8 -- August 1977

- Advanced Programming Techniques III: Sorting And Searching
- SR-52 Program: Ordered Hashing (Knuth Algorithms B & F)
- 58/59 Tips
    - Absolute Addressing
    - Op 18 and 19
    - More on Dsz
    - Getting the Correct Memory Partitioning
    - Neutralizing a Label
    - Out-of-Range Indirect Addressing
    - Neutral Error Producer
    - Pause Loop During Error Condition
- Friendly Competition
- General Purpose Plotter (59/PC-100A)
    - TI-59/PC-100A Program: General Purpose Plotter for y=f(x)
- More on y^x
- Execution of Unintended CROM Code (58/59)
- Trivia Award (52)
- Conversion of SR-52 Programs to 59ese
- 58/59 Relative Addressing

Volume 2 Number 9 -- September 1977

- HIR Operations: A Major Discovery (58/59)

- TI-59/PC-100A Program: Biorhythm Analysis
- Printer Head Cleaning (PC-100/PC-100A)
- Magnetic Cards (59)
    - Exploring Mag Card Protection
    - Mis -reads -writes
    - Mag Card Read and Write Under Program Control
    - SR-52 Cards for the TI-59
- A PC-100A Typewriter (58/59)
    - TI-58/59 Program: PC-100A Typewriter
- Membership Address Changes
- Miscellany
    - How the PPCs Work
    - TI Trig Algorithms
    - Owner's Manual Versions (58/59)
    - Numerical Solutions To Partial Differential Equations
    - More on INV and Ind Viability (52)
    - TI Notes

Volume 2 Number 10 -- October 1977

- Friendly Competition
    - SR-56 Program: Satellite Predictor
    - SR-52 Program: Extended Precision Powers
- Routines
    - A Short Flag-Reversal (52)
    - More on Sum of the Digits (V2N7p4)
    - Automatic Number Printer (58/59)
    - Factorials
    - Special Case Routines
    - Eigenvalue Calculations (58/59)
- I/O Ideas
- List/Trace Options Under Program Control And More On Printer Connection Sensing (58/59/PC-100A)
- Tips and Miscellany
    - No-Hassle Partitioning (59)
    - Clerical Aids
    - Statistics Keys Tricks
    - Extended Print Code (58/59)
    - More on Joel Pitcairn's Trig Algorithms (V2N9p6)
    - More on Printer Head Cleaning (V2N9p3)
    - CROM Library Notes (58/59)
    - PC-100 Modification for TI-58/59 Use
    - SR-52 Mechanizations of Analytic Models
    - Membership Address Changes
    - More on Card Road Under Program Control (59)
    - Ops 20-39 With a 959.0 Partition (58/59)
- Tic Tac Toe (59/PC-100A)
    - TI-59/PC-100A Program: Tic Tac Toe
- More on HIR Operations (58/59)

Volume 2 Number 11 -- November 1977

- CROM Use Enhancements (58/59)
    - Hyperbolic Trig Functions
    - Register Clearing Routine Extension

    - Bypassing Stores
    - Year Day with ML-20
- PPC Cryptology
- Advanced Programming Techniques IV: Topological Sorting
    - TI-59/PC-100A Program: Topological Sorting
- Some New SR-52 Tips and Discoveries
- Label Rules (58/59)
- Extending Data Memory via Efficient Packing
- Routines (58/59)
    - Unit Step and Delta Functions
    - Angle Mode Indicator
    - Digit Reversing (56,57,58,59)
- SR-56/TI-57 Program Exchange
- Membership Address Changes

Volume 2 Number 12 -- December 1977

- Routines
    - Double Precision (56,57,58,59)
    - The Mathematical Integer Function (56,57,58,59)
    - More Conditionals (56,57,58,59)
    - Automatic Number Printer (58,59) Revisited (V2N10p2)
- Membership Address Changes
- Tips and Miscellany
    - TI-58 Users Assistance
    - A p31 Application (52,58,59)
    - References to the 58/59 Owners Manual
    - CROM vs User Code Execution Speed (58,59)x
    - LRN Mode Lockout (58,59)
    - More on Dsz (58,59)
    - Ideas for a "Support Software" CROM (58,59)
    - More on Printer Head Cleaning
    - Protected Editing (58,59)
    - INV Viability (58,59)
    - Mag Card Printing (52,59)
    - SR-51/51A/PC-100A
    - SR-52 Extended Memory
    - HLT-R/S, rset-RST, and rtn-INVSBR 52-58/59 Differences
    - Fractured Display (58/59)
    - Corrections
    - Decapower Zero Suppression (52)
    - More on INV' and Dummy operations (56,58,59)
    - Program/Routine Listings for TI-56,57,58,59 Use
    - SR-52A Zero Divide
    - Flashing Display Variations
    - More on Execution of Unintended CROM Code (58,59)
    - Another SR-52 Analytic Model (V2N10p5)
    - Quaternion Arithmetic (59)
- Friendly Competition
- Keying Printer Character Code (58/59)
- Editorial: 52-NOTES Style/Club Philosophy Revisited

Volume 3 Number 1 -- January 1978

- Fractured Digits (58/59/PC-100A)
- Designing a Program/Data Comparator (59)
- Routines
    - Double Precision Multiplication (56,57,58,59)
    - Efficient Polynomial Evaluator (58/59)
    - Regulated Clock (58/59)
    - Fractured Digits Synthesizer (52)
    - Last Digits Viewer (52)
    - A Short Closed-Form Fibonacci Number Generator (56,57,58,59)
    - Error State Reversal (52)
- Tips and Miscellany
    - Friendly Competition
    - Strange CROM behavior (58/59)
    - Magnetic Card Extraction (59)
    - Custom CROMs (58/59)
    - Printer "Hiccup" (58/59/PC-100A)
    - Soft Flash (58/59)
    - A University Association of Calculator Programmers
    - Successful Mag Card Recording Check (59)
    - Agriculture Programs
    - Interrupt Processing (58/59)
    - Correction
    - CROM HIRs (58/59)
    - More on Unintended CROM Code Execution
    - Strange LRN Behavior (58/59)
    - Printer Character Shifting Using the Fix Function (58/59)
    - Writing Good Diagnostic Programs
    - Membership Address Changes
    - More on Out-of-Bounds Pointer Behavior (58/59)
- Fractured Digits Revisited (58/59)

Volume 3 Number 2 -- February 1978

- Practical Application of Obscure but Powerful Programming Features
- Efficient Print Code Storage
- Recording Current Machine States (52,59)
- Sorting and Searching: Some Popular Applications (59)
    - Addressing Block-Stored Data by Arbitrary Tags
    - Sorting Data By Magnitude
    - TI-59/PC-100A Program: Shell Sorting of up to 99 Data
    - TI-59/PC-100A Program: Bridge Deal
- Tips and Miscellany
    - A Safe, Handy Hardware Interrupt (58/59)
    - Use of p41 to Make Lbl Lbl Tricks Work for the 58/59
    - PC-100 to PC-100A
    - 56/57/58 Program Exchange
    - CROM HIRs (V3N1p5)

Volume 3 Number 3 -- March 1978

- Book Review: User Survival Guide for TI-58/59 Master Library
- A Review of the TI-57 Vis-A-Vis its Contemporary PPCs
- Fractured Digits: More Control (58/59)
- Label Search Methods: A Comparison of TI/HP Machines
- Routines

        - Fibonacci Number Generator Revisited (V3N1p3)
        - Efficient I/O Handling For Engineering Problems (58/59)
        - Copying the Display into the T Register (56,57,58,59)
        - Digit Reversing (V2N11p6)
    - Tips and Miscellany
        - A New Hidden Facet of the Old Machines (52,56)
        - Extra Memory for the TI-58
        - Data Entry Sensing (V2N7p5)
        - A New R-P Convention (57,58,59)
        - Fast Constants
        - Runaway Mag Card Drive Motor (59)
        - More on Pgm mm R/S (V3N1p3)
        - Creating Large Mantissas (57,58,59)
        - Overflow and Underflow (57,58,59)
        - Membership Address Changes
        - Absolute Code Execution Speed (52,56,58,59)
        - HIR Operator Modifications (58/59)
        - More on CROM HIRs (V3N2p6)
        - Mag Card Tips
        - More on the Hardware Interrupt (V3N2p5,6)
        - CROM Pause
        - DC Adapter
        - Quick Manual Flag Tester (58/59)
        - The R/S' Function (58)
        - CLR/CLR' Difference (58/59)

Volume 3 Number 4 -- April 1978

- Detecting Machine Hardware Changes
- Precision and Accuracy
- Designing a Practical File Manager (59/PC-100A)
    - TI-59/PC-100A Program: File Manager
- Advanced Programming Techniques V: Designing Operating Systems
    - TI-59/PC-100A Program: BASIC Operating System Simulator
- Book Review: Programmable Calculators
- Tips and Miscellany
    - More on Strange LRN Behavior (V3N1p5)
    - Local Club
    - Membership Address Changes
    - Printer Spacing (58/59/PC-100A)
    - A Tic-Tac-Toe Option (V2N10p6)
    - More on Fractured Display (V2N12p3)
    - Use of Contributed Material
    - Correction (V3N3p6)
    - Used SR-52s
    - Merged Code Labels (58/59)

Volume 3 Number 5 -- May 1978

- An Analysis and Comparison of 3 Calendar Programs (59/PC-100A)
    - TI-59/PC-100A Program: Calendar Printer
    - TI-59/PC-100A Program: Calendar Printer
    - TI-59/PC-100A Program: Calendar Printer
- Program/Routine Copyright
- 52-NOTES Subject Index

- Periodical Review: TI-59 Newsletter
- Tips and Miscellany
    - 56/57/58 Program Exchange
    - CROM Download into Used User Memory
    - TI Service Manuals and Parts
    - More on Memory Add-On (V3N3p4)
    - EE and Eng (57,58,59)
    - CROM Use of Flags for Print Suppression

Volume 3 Number 6 -- June 1978

- Friendly Competition
    - TI-59/PC-100A Program: Calendar Printer
    - TI-59/PC-100A Program: Calendar Printer
- Book Review: Programmable Calculators - Business Applications
- Tips and Miscellany
    - Membership Address Changes
    - CROM RST Behavior
    - Structural and Applied Mechanics PPC Applications
    - Last-Step Instruction Execution under Programmed List (58/59/PC)
    - Subroutine Execution from the Printer (52,56,58,59,PC)
    - An All-Digits Generator
    - Strange Pause Effects (58/59)
    - A Search for Useful CROM Dsz Loops (58/59)
    - Shortened Paper Drive (PC-100/100A)
    - More on Runaway Motor (V3N5p5)
    - Linear Programming
    - EE on a 0-digit Mantissa (58/59)
    - A Manual SBR Quirk (58/59)
    - Computer Listings of SR-52 Programs
    - CROM Partitioning
    - ML-24/25 Precision
    - Card-Drive Adjustments (59)
    - CLR for INV EE
- Routines
    - CROM-Call Selective Trace (58/59/PC)
    - A General Purpose Integer Routine (V2N12p1)

Volume 3 Number 7 -- July 1978

- Machine Number Scaling Terminology and Display Format Notation
- Special-case Search Programs
    - TI-58/59 (PC) Program: Solutions to $a^3+b^3+c^3=100a+10b+c$
- Code Synthesis via Op 8 Label Listing (58/59/PC)
- Calendar Printing Competition (continued)
    - TI-59/PC-100A Program: Calendar Printer
- Routines
    - More on the Mathematical Integer Function (V3N6p6)
    - A Self-calling Subroutine (V3N2p1)
    - A Short INV Int X 100 = Replacement (58/59)
    - Print Code References (58/59/PC)
    - Improved Open Parentheses Counter (58/59)
    - Selective Register Listing (58/59/PC)
    - Hyperbolic Trig Functions Shortcut (V2N7p3)
    - A Shell-Sort Application (V3N2p5)

- Tips and Miscellany
    - Detecting Machine Hardware Changes (V3N4p1)
    - SR-56 Pseudos
    - Printer-Connection Slowdown (58/59/PC)
    - A Totally Blank Display (58/59)
    - Club Support of TI PPCs
    - 56/57/58 Program Exchange
    - PC Carrying Case

Volume 3 Number 8 -- August 1978

- Suppressed Operand Instructions (57)
- Language Synthesis (58/59/PC)
    - TI-(58)/59/PC Program: First Order English Synthesizer
- Membership Address Changes
- Special case Processing (continued)
    - TI-59/PC Program: Solutions to $a^3+b^3+c^3= 100a+10b+c$
- Tips and Miscellany
    - An Update on TI CROMs
    - An Update on TI's Customer Relations Telephone Service
    - Battery Pack Interchangability
    - Machine Reaction to High Humidity
    - Club Program Exchange
    - More on Dummy Operations (V2N12p3)
    - 10 y^x Vs INV log
    - Calendar Printer Status
    - An R/S or rtn Quirk (58/59)

Volume 3 Number 9 -- September 1978

- Roots of Polynomials
    - TI-59/(PC-100A) Program: Quadratic, Cubic Quartic Root-Finder
- Calendar Printer Competition (continued)
    - TI-59/PC Program: Calendar Printer
- Roman Numeral Programs
    - TI-59/PC Program: Roman Numerals
- Tips and Miscellany
    - An effective Op3mn on 2-Digit Registers (58/59)
    - Efficient Number Base Conversions (52,57,58,59)
    - Clearing Program Registers (52)
    - Programming a Variable EE Function (52)
    - CROM Availability Abroad
    - PC-100B
    - Club Program Exchange
    - 58/59 Service Manuals
    - More on Printer Spacing (V3N4p6) and other Behavior
    - Mag Card Read/Write Error Minimization (59)
    - Blank Display (V3N7p6)
    - Membership Address Changes
    - Correction (V3N7p5)

Volume 3 Number 10 -- October 1978

- The Business Decisions (9) CROM
- Bridge Deal (V3N2p5)
    - TI-59/PC Program: Bridge Deal

- Print Buffer Filling with an EE or Eng Display (58/59/PC)
- Last Digit Viewers
- Some 58/59 Firmware Revealed
- Repeating Decimals
- Tips and Miscellany
    - The Math Integer Function (V2N12p1, V3N6p6, V3N7p5)
    - TI's Sourcebook for Programmable Calculators
    - Flag Status Routines (52,58,59)
    - The Newest 59s
    - Membership Address Changes
    - Machine Power Consumption (Blank Display V3N9p6)
    - Writing HIR Sequences (58,59)
    - Flashing Display Variations (V2N12p4)
    - Mag Card Cues (52,59)

Volume 3 Number 11 -- November 1978

- More on the Revealed Firmware (V3N10p4)
- Fast Number Base Conversion Routines (V3N9p5)
- Friendly Competition
    - TI-58/59/PC Program: Speedy Factor Finder
- Repeating Decimals (V3N10p7)
- Tips and Miscellany
    - Calculator RF Interference
    - Used SR-52s and SR-56s
    - Oil and Gas Well Accounting and Taxes
    - PC Battery Charging
    - Games Buffs
    - PC Roller Cleaning
    - More on Op3mn on 2-Digit Registers (V3N9p5)
    - CROM Calls to Undefined Labels
    - Membership Address Changes

Volume 3 Number 12 -- December 1978

- Roots of Polynomials (V3N9p1)
    - TI-59/PC Program: Polynomial Roots
    - TI-59/(PC) Program: Quadratic, Cubic, Quartic Roots
    - TI-59 Program: Quartic, Cubic, Quadratic Solutions
- The Math/Utilities CROM (V3N8p5,6)
- Revealed Firmware (V3N10p4, V3N11p1,2)
- Two New Periodicals
- Tips and Miscellany
    - Membership Address Changes
    - The Washington (DC) Area Local Club

Volume 4 Number 1 -- January 1979

- Off-the-Shelf Custom CROMs
- The Game of Life
    - TI-59/PC Program: LIFE
- More on Program Copyright (V3N5p4,5)
- Book Review: Sourcebook for Programmable Calculators
- CROM RNG Limitations
- The PC-100C Printer
- A CROM Op Ind

- Tips and Miscellany
    - Printing 12 Places (58/59/PC)
    - Revealed ROM Constants (V3N12p5)
    - Friendly competition (V3N12p6)
    - Bridge Deal (V3N10p2)
    - A CROM Identifier
    - Membership Address Changes
    - An INV HIR Application
    - Print Overlays (PC)

Volume 4 Number 2 -- February 1979

- Transparent CROM Checks (58/59/PC)
- More on CROM RNG Limitations (V4N1p4)
- Special Case Processing (V3N8p5)
    - TI-58/59/PC Program: Solutions to $a^3+b^3+c^3=100a+10b+c$
- Efficient Data Packing (V2N11p5)
- Editorial: Looking Ahead
- Interpolation and Extrapolation
    - TI-59(PC) Program: Polynomial Least Squares Fit
    - TI-59(PC) Program: Variable Spacing Interpolation and Extrapolation
- Tips and Miscellany
    - Off-the-Shelf Custom CROMs (V4N1p1)
    - Euclid's Algorithm Routine (V3N11p5)
    - Forced Card-Read(59)
    - Friendly competition (V4N1p6)
    - Membership Address Changes
    - Correction (V3N12p2)
    - Print Borders(58/59/PC)

Volume 4 Number 3 -- March-May 1979

- A Keyboard SBR N in One Step (58/59)
    - TI-59/PC Program: Fast Typewriter
- Vectored Processing with Op 10
- Book Review: Countdown: Skydiver, Rocket and Satellite Motion on Programmable Calculators
- More on the MU CROM (V3N12p3-5)
- *** David W Johnston (5) -1979 ***
- Zero as a Callable Label (58/59)
- ROM Constants Update (V4N1p6)
- Corrections
- Membership Address Changes