52-NOTES

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 routine call). 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, 11-578, 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.

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 sanc tioned by Texas Instruments, Inc. Membership is open to any interested person: $6.00 ($10.00 US abroad) includes 6 issues of 52-NOTES; back issues start June 1976 @ $1.00 each ($1.67 abroad).

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 succeeding 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 $(ax_n+c)\mathrm{mod}\ m$ is an integer before it is used to generate x_{n+1}. This is accomplished by putting $\mathrm{Int}(r_9+\frac{1}{2})$ into Reg 9 before calling SBR D.MS (A for MU-12). (r_9 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: LA R10 X R9 + R11 = ÷ R12 = INV Int X xXt R12 + .5 = Int S9 xXt rtn, and run with r_{10}=24298, r_{11}=99991, and r_{12}=199017. But, ofcourse, 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. Michael 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 Van Wye (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 $a^3+b^3+c^3=100a+10b+c$ John VanWye (982)

User Instructions: Run by pressing A; results in 69 seconds.

Program Listing:

```
000:  R*2 + R*3 = xXt 336 ± xGEt 046 210 ± xGEt 046 120 ± xGEt 046 60
030:  ± xGEt 046 24 ± xGEt 046 6 ± xGEt 046 0 x=t 076 2 SUM2 Dsz 0 000
056:  4 S00 8 INV SUM02 2 SUM3 Dsz 1 000 Ifflg0 112 Adv R/S ± yx 3 1/x
080:  + 10 X (R2 - 4) + 100 X (R3 - 12 = Prt S22 0 x=t 146 GTO 049 4
113:  S00 5 S1 S02 13 S3 Rst LA 4 SQ S02 5 S01 12 S3 Stflg0 Fix0 GTO
144:  000 R22 + 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 problems 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 xXt, 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 xXt, 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:  LA S1 rtn ((CE - 1)÷ R1 S0 Int INV SUM0) S2 (INV Int X R1 Int)
031:  S3 Op23 4 SUM2 R*2 INV Int S*2 (INV Dsz3 067 (R0 X 10) S0 Int
060:  INV SUM0 + GTO 045 0) INV Log D.MS S3 P*2 rtn LB SBR005 R*2 Int
085:  INV SUM*2 (Exc0 X 10) Int INV Log DMS Prd0 Prd*2 Prd3 ((1/x X xXt
110:  ) (INV Int ÷ xXt) Int + Exc0 + R*2 INV Int) S*2 R3 INV P*2 R0
136:  rtn LC SBR005 (R*2 INV Int X R3 INV P*2 (R0 X 10) Int INV Log DMS)
165:  Int rtn LD SBR005 R*2 Int INV SM*2 (Exc0 X 10) Int INV Log DMS P0
191:  P3 P*2 (1/x X xXt) (INV Int ÷ xXt) ((Int + R*2 INV Int + R0) ÷
219:  R3) Exc*2 Int rtn   Note:  Pn=Prdn and P*n=Prd*n, n=0,1, ...9
```

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 foreseeable 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   Bill Skillman (710)

User Instructions: Key order n (less than 7), display flashes if ML module is not connected; press A; key x,y pairs: xi, press B, yi, 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 a0, a1, ...an. To interpolate or extrapolate, key x, press E, see y displayed and/or printed.

Program Listing:

```
000:  LE Pgm7 C rtn LA Pgm19 SBR588 CMs S7 Prt X (CE + 5) + 11 = S00
029:  9 Op17 1 S8 rtn LB S2 Prt R0 S04 9 S5 R/S S3 Prt Sm*4 R7 S6 S01
060:  1 X R2 X SM*5 xXt Op25 R3 = Op34 SM*4 xXt Dsz6 061 X R2 = SM*5
088:  Op25 Dsz1 082 Op28 R8 rtn LC Op38 R7 + 8 + S1 R7 S4 S2 x^2 =
118:  SM2 R7 SM1 S3 Op23 R*1 S*2 Op31 Op32 Dsz3 128 Dsz4 120 Op27 Pgm2
148:  C R0 S4 R7 S2 1 Pgm2 D R7 x^2 + 7 = S1 R5 xXt Op21 R*1 INV x=t
178:  172 R7 SM1 Op25 0 Exc*4 S*1 Op34 Dsz2 161 Pgm2 E 1 Pgm2 A' R7 S0
208:  S4 Op34 5 S2 Pgm2 R/S S*2 SBR236 Op22 Dsz0 215 6 Op17 Adv Adv
235:  rtn Op8 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 Interpolation and Extrapolation Ed

User Instructions: Key x1, press xXt, key y1, press E; key xi, press xXt, 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: LE CMs S21 S50 xXt S11 12 S00 22 S01 1 S02 44 Op4 R11 Op06 45 Op4
033: R21 Op6 L1' R/S S*1 xXt S*00 44 Op4 R*0 Op06 45 Op4 R*1 Op6 Op20
063: Op21 Op22 R2 GTO 1' LA 31 S4 R2 S5 R12 - R11 = S03 11 S0 L2' R*0
097: - R11 = ÷ R3 = S*4 Op24 Op20 Dsz5 2' R2 - 1 = S05 2 S8 R21 S41 L3'
130: 1 S52 0 S51 R8 - 1 S7 = S9 L4' R8 - R7 = S06 30 + R8 = S4 R*4 -
166: (30 + R6) S4 R*4 = Prd52 40 + R6 = S53 R*53 ÷ R52 = INV SM51 Op27
199: Dsz9 4' 20 + R8 = S53 R*53 ÷ R52 + R51 + (40 + R8) S53 0 = S*53
232: Op28 Dsz5 3' CLR Adv R/S LC xXt 67 Op4 xXt Op6 - R11 =÷R3 = S54
260: R41 S52 2 S6 R2 - 1 = S5 L5' 1 S52 R6 - 1 = S09 1 S7 L6' R54 - (30
297: + R7) S4 R*4 = Prd52 Op27 Dsz9 6' 40 + R6 = S53 R*53 X R52 = SM51
329: Op26 Dsz5 5' 45 Op4 R51 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 if 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 xXy 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.