52-NOTES

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 as integers in 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 in 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 manipulations 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(a1,a2,...an)=g(a1,a2,...am), for which given different functions f and g, it is desired to determine what values of a1,a2,... 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: LA R*12 + R*13 + R*14 = xXt R11 x=t 040 Dsz11 022 R/S Dsz14 A
025: 10 S14 Dsz13 A 10 S13 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: $a1^n+a2^n+\ldots$ $an^n=10^{n-1}a1+10^{n-2}a2+\ldots$ an, although even the fastest computers would be too slow for very large n, unless significant 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: $a1^n+a2^n+\ldots$ $an^n = b^{n-1}a1+b^{n-2}a2+\ldots$ an where b is any number base, that for b=2, there are only the trivial solutions n=1, and ai=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), I briefly 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: LE 10 Op 17 CMs 92 S90 76760869 S99 85760576 S98 32766676 S97 1005766176 S96 9 Op 17 SBR 160 Adv Adv Adv R/S. If actually used with a 57, only the printed mnemonics apply: LBL 5 + PAU xXt GTO 5.

CALENDAR PRINTER 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 in 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:  LA S1 S4 S5 - 13 = ± S47 8 Op17 67 SUM04 2 S45 R5 R/S H4 R/S
031:  LB 0 S47 LC H14 S3 Pgm20 SBR086 Pgm20 SBR177 GTO 066 7 xXt R1
059:  INV xGEt 066 - 7 = S1 ± S00 12 SUM0 CP R*4 H8 Int INV x=t 125
085:  3 S45 H14 ÷ 4 = INV Int INV x=t 124 H14 ÷ 400 = INV Int x=t 123
111:  H14 ÷ 100 = Inv Int x=t 124 1. SUM1 - 15 = ± S03 4 Prd3 S44 H14
141:  Op6 R*3 S42 Op23 R*3 S41 R64 Op1 R65 Op2 R66 Op3 R67 Op4 Op5
171:  INV Dsz45 188 Op23 R*3 S40 Op23 R*3 S39 R*0 SUM0 H5 R*0 + 99 =
200:  H6 Op20 R*0 SUM0 - 1.99 = H7 R*0 SUM0 H8 Op5 Dsz44 188 Op0 R*0
232:  SUM0 H5 R*0 + 99 = H6 Op20 R*0 x=t 278 SUM0 - 1.99= H7 R*0 SUM0
265:  H8 Op5 Op0 R*0 x=t 280 H5 Op5 Adv Op25 Op24 Dsz47 055 CLR rtn
291:  LD S46 C Adv 12 INV SUM4 S47 1 H34 S5 S1 H14 xXt R46 xGEt 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 0 2.000402 2.000401000402
50: 2.000312000401 2.000311000312 0 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 xGEt 1' + INV Int x=t 1' 1 ± - L1' = 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: LA seq1 B seq2 A seq3 R/S LB 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: LA 1 Prt B 2 Prt A 3 Prt rtn LB 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: LA 8 S0 S1 CLR LA' + 0p4 0p6 1 Dsz0 A' 8 S00 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 0p4 0p06 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: LA xXt 10 S1 L1' (INV Stflg 7 0p31 0p22 0p18 Ifflg 7 1' CE L2' Dsz2 3' xXt rtn L3') 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 a0 is the first and Reg bb0-1 the last: LA (xXt 0p16 INV Int X 10 + 1) ((xXt X (INV Int X 100) 0p17 1) Int X 10) INV Lst 0p17 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 cells 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 turn-on): 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 converning 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.

Storing 13 digit numbers

15.00012000201

Key, 1200021 ÷ 1 EE 11=STO n 15 SUM n

-25.03633131617

Key, 3633131617 +/- ÷ 1 EE 11=STO n 25 +/- SUM n

-.1234567891234

Key, 456789 +/- ÷ 1 EE 13=STO n .123 +/- SUM n

11.124596403

Key, 124596403 ÷ 1 EE 9=STO n 11 SUM n
Or key, 11 + .124596403=STO n

2.010401000402

Key, 401000402 ÷ 1 EE 12=STO n 2.010 Sum n

2.123456789123

Key, 3456789123 ÷ 1 EE 12=STO n 2.12 SUM n