52-NOTES

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 corresponging 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 is 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 arrives at this conclusion by suggesting "... that Op 12, Op 14, and Op15 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 $\overline{x}$ 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 could 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, $\overline{x}$ 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 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: LA S0 CP CLR R0 L1' ÷ 8 - Int S0 = X 8 = Pause R0 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: LB X 8 - 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: .07715335156345432 ... .

Generalizing these routines to handle other number bases in addition to octal is easy enough, and does not slow execution speed perceptably. For example: LE S1 R/S LD CP S0 L1' R0 ÷ R1 - Int S0 = X R1 = Pause R0 INV x=t 1' R/S L2' X R1 - Int Pause = GTO 2' LB 1 S0 CLR R/S L3' X R0 + R1 Prd00 0 R/S GTO 3' LB' R1 S0 CLR R/S L4' ÷ R0 + R1 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 equivalent 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 as 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: R2 R3 ÷ Frac x=0 GSB 0 RC S+3, which takes about .38 seconds to execute. Corresponding 59 code looks like: R2 ÷ R3 = 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 3,3,1717,379721. 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 LC SUM2 R1 ÷ R2 = INV Int x=t 2' rtn L3' CP 6 C 4 C 2 C 4 C
027:  2 C 4 C 6 C 2 C R2 xXt R3 xGEt 3' R1 Prt Adv Adv Adv CLR RST
051:  L2' R1 ÷ R2 Prt = S1 √x S03 0 GTO C LE CP S1 Prt Adv √x S03 1
079:  S02 1 C 1 C 2 C 1 S02 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 repeating part plus the non-repeating part, the two expressed as rational fractions and added; thus .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...900...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, .8652597402597402... 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: LA CP S1 ÷ R/S S2 L1' = INV Int x=t 2' X R2 S1 = EE INV EE S2 R1 ÷ R2 GTO 1' L2' R2 R/S.

But if a numerator or denominator is toobig 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 1359/1571: 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: LA S2 L5' S04 0 S3 L1' R1 xXt R3 + R4 = ÷ 2 = Int S5 ÷ R2 = xGEt
033: 2' R5 - Exc3 = CP x=t 3' GTO 1' L2' x=t 4' R3 xXt R5 x=t 3' S4
058: GTO 1' L3' Op22 R2 GTO 5' L4' R5 Prt xXt R2 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 ... 261146496815 2 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-51s 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 Op 3mn 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.