52-NOTES

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: ...S2 Pgm 5 C'... for Sinh, and ...S2 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 (688) 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): LA S1 R/S S2 R/S S3 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 relationships 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 so-called link fields: a part of each element (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 LT j,k LT 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 ($j \neq k$) 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: LE xXt 9 Op17 xXt CMs S34 39 S37 31 Op4 R34 Op6 Adv 2431 Op4 CLR
030: R/S Op6 - Int S32 = X 100 = S33 + R*37 INV Int = S*37 1 SUM*33
056: R*32 INV Int + R*37 INT = S*37 R37 ÷ 100 + R*32 Int = S*32 1 SUM
082: 37 SUM 38 R38 GTO 030 LA Adv Adv 324137 Op04 0 S38 1 SUM38 CP
109: R*38 Int INV x=t 130 R38 + R*35 INV Int = S*35 R38 S35 R34 xXt
133: R38 INV x=t 105 R0 S36 R36 CP x=t 221 Op06 1 INV SUM34 R*36 INV
158: Int X 100 = S37 CP x=t 213 R*37 S31 1 INV SUM*31 R*31 Int INV
182: x=t 199 R*37 Int + R*35 INV Int = S*35 Int S35 R31 INV Int X
204: 100 = S37 GTO 166 R*36 Int S36 GTO 143 Adv 31351730 Op4 R34 Op6
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: LA 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: LA ifpos S9, and steps 5-8 are skipped when the display is positive (though ofcourse 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 separater; 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: LA + 12 Prd 01 0 = SUM 1 rtn LB R1 ÷ 12 - Int S1 = 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: LA 12 S98 L1' R/S + 12 Prd*99 0 = SUM*99 Dsz "98" 1' 1 SUM 99 GTO A LB 12 S98 L2' R*99 ÷ 12 - Int S*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/S, 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 1 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 LT 0, H(x)=1 if x GE 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: LA 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: LA (CE ÷ 10 Prd 1 - Int S2) SUM 1 R2 INV x=t A (Exc 1 X 10) rtn, where for the 56 and 57, LA 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 88001; 697: 21 Kristin Dr #718 Schaumburg, IL 60195.