52-NOTES

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: L1 123 L2 45678 R/S L3 GTO:: 2 L4 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 "bare" 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

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.

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 quasi-intelligence 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.

TI-(58/59/PC Program: First Order English Synthesizer Ed

```
000:  LB R57 X π = INV Int S57 X R59 = xXt 28 S0 L1' Op20 R*0 + INV GE
029:  1' CLR 28 INV SUM0 R*0 rtn LA S57 S58 36171716 Op4 R58 Op06 7 Op17
062:  L2' 4 S61 L3' B S60 B EE 2 SUM60 CLR B EE 4 SUM60 CLR B EE 6
087:  SUM60 CLR B EE 8 + R60 = .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 28, 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 occuring 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. Bennett'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 a random number in the 1 to $B_i + B_{i+1} + \ldots B_{i+20}$ range.
4. Pack P_k in the next print buffer position; print a 20-character line; change the value of i to 21k + 1.
5. 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:  8,0,1,1,0,0,0,0,10,1,0,0,0,3,0,0,0,1,0,0,0,24,0,0,1,0,0,0,0,48,63,
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 $B_i + B_{i+1} + \ldots B_{1+20}$ 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 challanging. 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 2 challengers to the V3N7p3 program both reduced execution time by more than half, with Bill Skillman (710) leading John Mickelsen (990) 7 min 41 sec to 8 min 23 sec with a revision to my V3N7p3 program. Incidentally, 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 a sort of 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 $a^3+b^3+c^3=100a+10b+c$ Bill Skillman (710)

```
000:  0 GTO 109 Nop Nop Nop R*21 + R*22 = xXt 384 INV GE 078 0 GE 075
025:  99 xGEt 075 171 GE 075 192 GE 075 273 GE 075 288 GE 075 336 GE
058:  075 357 GE 075 375 GE 075 384 x=t 103 Dsz21 007 10 S21 1 SUM22
089:  Dsz23 007 Adv R/S   099:  1 GTO 001 S24 GTO*24   109:  X 100 +
114:  R21 X 10 + R22 - 21 = Prt GTO 078 LA 10 S21 S23 11 S22 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

<u>An Update On TI CROMS</u>: 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 on 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 transfered 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 programs. So Exchange users are asked to be patient, and wish Dave lots of luck!

More On Dummy Operations (V2N12p3): Dick Blayney (610) notes that a 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 + S7 = 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 $x=\pm67$, and for INV log at $x=\pm95$. For this experiment, the exact solutions are easily generated via 1 EE x, or multiples 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.