52-NOTES

Volume 3 Number 4 48/39/38 April 1978

Newsletter of the SR-52 Users Club
published at
9459 Taylorsville Road
Dayton, OH 45424

DETECTING MACHINE HARDWARE CHANGES

As was the case with the 52 and 56, TI is making hardware changes in the new machines without changing names or adding revision identifiers. Fortunately, the date of manufacture is stamped on the back of each machine. (As I recall, one of you phoned this information to me, but I neglected to write your name down). By relating machine characteristics to date made, perhaps we can determine when detectable mods were made. The date is coded in 4 digits following the letters LTA (DTA for older machines made in Dallas rather than in Lubbock) in an aabb format, where aa is the week and bb the year. If one of you electronics-hardware experts will volunteer to be the focal point, I'll invite members to describe their machines (operational behavior, component descriptions, PC board design, etc) to you, and publish your findings. Dave Leising (890) notes a component difference in late-model 59s which he identifies with a card read/write improvement. I confirm Dave's identification of an added transistor, and note also the addition of more resistor/capacitor-like components, and a different PC board on a 59 made the 49th week of 1977, as compared with one made the 24th week. The newer 59 does indeed read and write mag cards with fewer errors, and displays a brighter C when calculating (which may not be an improvement if this requires more power).

The ability to relate machine configuration to the date made should help the buyer find the best machine design currently available, as well as give the user a better understanding of his particular machine's behavior.

PRECISION AND ACCURACY

In the context of PPC calculations, precision is the degree to which the machine representation of a number approximates an absolute definition of that number; accuracy is the degree to which calculated results approximate an absolute definition of the results. Thus while precision contributes to accuracy, it is not the only factor, and how calculations are produced may affect accuracy even more than precision. We have already confronted cases where trig function processing can cause problems (V2N2p1 and V2N5p4), and there are many cases where truncation at the LSD causes grief. Professor W Kahan at UCLA (Berkeley) examines machine accuracy versus apparent precision in a recent memo (of limited circulation), introducing some concepts which users may find helpful in the analysis of calculated results. He states that while the easiest way for a user to find out what's happening is via a so-called Forward Error Analysis (FEA), what must usually be made is a Backwards Error Analysis (BEA). If a function f(x) is approximated with machine processing by F(x), then F(x) submits to a satisfactory FEA when F(x)=almost f(x). But this may only be true when the x specified in f(x) is exactly the same as the x used in F(x). What usually happens is that F(x)=almost f(almost x) which can easily cause F(x) to be nowhere near f(x), and is a situation which requires BEA according to Kahan. Keeping in mind the 2 almosts and their possible cumulative effect should help the user understand what's going on, whether he approaches error analysis quantitatively or just qualitatively. Members are invited to further explore FEAs and BEAs vis-a-vis the PPCs.

DESIGNING A PRACTICAL FILE MANAGER (59/PC-100A)

While it is tempting to try to devise ingenious search techniques (V3N2p3) to minimize the number of data registers required for arbitrary-key access, it may turn out that for most practical applications execution speed is more important than register economy.

The following program ties up all possible data registers, but is fast and easy to use. Members are invited to try it (or their own versions) in real-world applications and to report results, suggest improvements, etc. Files are organized by sets of 2 mag cards, each pair holding up to 99 files whose keys are 1 or 2-digit numbers in the 1-99 range. File contents (records) are interpreted as data when in the 10^{-99} to 10^9 range, and as character strings when GE 10^9. Provision is made to assign an identifying number to each card-pair, and this tag along with all stored records is preserved as a card-pair is read in from time to time for file reconfiguration. Files may be accessed in any order for store, recall, add, or subtract operations, and a listing made at any time of all non-zero records. A printed record is made of all transactions, and users reconfiguring files many times may wish to include revision information in a card-pair identifier. For example 25.12 might identify the 12th revision of card-set 25. This tag is mag-card-stored in Reg 0, but preserved in H4 during file reconfiguration. Reserve an otherwise blank card to serve as a master for transfering the program to side 1 of each new card-pair, keeping in mind that steps 160-239 are treated as data.

TI-59/PC-100A Program: File Manager Ed

User Instructions:
1. To Prepare A File Manager Master Card:
   a. Write the program listed below into blank user memory.
   b. In RUN mode key 10 Op 17 1 2nd Write.
   c. Insert card, and record.
   d. Mark card: "File Manager Master".
2. To Prepare For Generating a New File Block:
   a. In RUN mode key 10 Op 17 CLR.
   b. Insert the File Manager Master Card, and read it.
   c. Key card number (any real), press 2nd E'.
3. File Processing:
   a. To Address a File: Key its 1 or 2-digit key, press A, see its contents displayed.
   b. To store a Record: Key the value (numbers GE 10^9 are interpreted as character strings). If 3a was just performed, press R/S. Else press SBR STO.

3. c. To recall a Record: Press SBR RCL.
   d. To Add to a Record: Key value, press SBR SUM.
   e. To Subtract From a Record: Key value, press ± SBR SUM.
   f. To List all non-zero Files: Press SBR List.
   g. To Record Current File Configuration: Press E, or for new or revised card ID, key ID, press 2nd E'; record all 4 banks on 2 mag cards.
4. To Address an Old File Block:
   a. Read the 4 card-sides into memory, with a 159.99 partition.
   b. Press C and do step 3 as desired.

Program Listing:

```
LA S0 LD 261745 Op4 R0 Adv Op6 R*0 rtn LS S*00 363732 LB Op04 1 EE 9
039:  xXt CLR R*0 xGEt 049 Op6 rtn Op2 Op5 rtn LR 351527 GTO B LSUM
066:  xXt 364130 Op4 xXt SUM*0 Op6 rtn LList Op00 27243637 Op2 Adv
096:  Op05 99 S0 CP R*0 x=t 118 D 35153516 B Dsz0 102 Adv Adv Adv Clr
126:  R/S LE' H4 31 LE 153516 Op4 H14 S0 Op6 GTO 122 LC R0 H4 GTO E
```

- - - - - - - - - - -

ADVANCED PROGRAMMING TECHNIQUES V: DESIGNING OPERATING SYSTEMS

Computer operating systems (OS) generally consist of a control or executive program, and the programs it directs to manage the required communication between mainframe computers, their periferal devices, and users... utilities such as compilers, assemblers, loaders, editors, file maintainers, interrupt processors, job schedulers, job control language interpreters, etc. Among these utilities, one of the more challenging to design is a compiler, which translates a high order language (HOL) into assembly (AL), or machine language (ML). The difficulty lies mainly in the interface between how a human can best express what he wants a program to do, and how a machine can be made to implement his intent. Humans tend to think most effectively using concept-linked symbol strings, and HOLs (FORTRAN, BASIC, COBOL, APL, etc) are designed to let the programmer use familiar symbols as "naturally" as possible. An AL is almost understandable to the machine: each instruction is symbolic, but has a one-to-one correspondence with a ML instruction (a binary number). An assembler translates AL code into ML code, and a loader puts the ML code into the proper memory locations.

PPCese has some of the characteristics of HOLs, ALs, and MLs, but there is not much in common with any one of these, So it's a challenge even with the powerful TI-59/PC-100A combination to simulate even just a part of a mainframe compiler and its associated OS support. The program which follows simulates the implementation of a BASIC programming construct via a remote terminal, The user "types" an assignment statement consisting of single-character variables and operators, following the usual convention that only a single variable appears to the left of the = sign. The usual + - * / arithmetic and ↑ exponent symbols are used, along with parenthetical nesting. Following input of up to 20 print codes (a variable may be any of the 59's 64 characters which is not one of the designated operators or parenthesis symbols), the program prints the statement as it would have been typed at a computer terminal, proceeds on into the interpretation process, calls for variable data to be input, runs the compiled/assembled/loaded code, and prints the answer. The user may then make more runs with the same program with new input data, get a listing of the program, or begin compilation of a new statement.

Code transfer rules (V1N2p5 and V3N1p2) complicate the process of synthesizing instructions as data, and in order to dodge position B restrictions, AB positions are always set to 60 (the code for DEG, which does no harm). Positions CD are also set to 60 in cases where RCL n would otherwise be split by a 60 at the AB position. The resulting compiled code is perhaps somewhat realistically inefficient, just as real compilations are usually less efficient than human-designed AL code. AOS architecture essentially eliminates the real-compiler requirement to translate HOL semantics into the proper ordering of machine instructions. Mechanizing this simulation on an RPN machine would be considerably more complex. There is program memory to spare if some register assignments are changed, and members are invited to add enhancements, or mechanize better approaches.

TI-59/PC-100A Program: BASIC Operating System Simulator Ed

User Instructions:
1. Initialize: Press E
2. Input BASIC Assignment Statement: Key the 2-digit print code for a character representing either a variable or an operator, press R/S. Repeat for up to 20 characters. The first character must be a variable, the second the = symbol (64).
3. Initiate Processing: Press A. (This step is done automatically following input of the 20th character). See the BASIC statement printed, followed a minute or so later by the cue: KEY n, where n is the character representing the first variable in the BASIC statement. Abort processing with R/S if the statement is incorrect as printed, and go to step 1.
4. Run The Compiled Program: Following each printed cue, key the data value for the indicated variable, press R/S. Following input of the last requested variable value, processing begins and the answer printed. For new inputs, press B, and repeat this step.
5. To Get a Listing of the Compiled Code: Press SBR List.
6. For New Compilation: Go to step 1.

Note: Record program with turn-on partition; program terminates with a 599.49 partition.

Program Listing:
```
000: LC' INV Stflg1 R*49 xXt 47 x=t 055 20 x=t 058 51 x=t 061 63 x=t
026: 064 60 x=t 067 55 x=t 070 56 x=t 073 xXt ± S*49 1 SUM48 R48 Stflg1
054: rtn 85 rtn 75 rtn 65 rtn 55 rtn 45 rtn 53 rtn 54 rtn LE' C' LB'
081: Ifflg0 165 X R46 = SUM*47 .01 Prd46 1 SUM49 Ifflg1 110 Dsz45 109
107: Stflg0 rtn INV Dsz45 135 R46 X 43 = SUM*47 .01 Prd46 Dsz45 134
132: Stflg0 rtn 4 EE ± 13 SUM*47 1 EE 37 LA' Prd*47 1 SUM47 .01 S46
158: 6 S45 INV Stflg0 rtn Ifflg1 204 xXt 1 SUM49 xXt ÷ 10 =.S43 Int EE
182: ± 13 SUM*47 R43 INV Int X 100 + 7 = INV Log EE GTO 148 xXt 6 EE ±
208: 13 SUM*47 1 EE 7 A' xXt GTO 081 LE 6 Op17 CMs 0 S0 Op00 20 S49
235: CLR R/S S*49 Op20 Dsz49 236 LA 1 S46 20 S49 R0 S47 5 S48 CLR X
263: 100 + R*49 = INV Dsz49 293 INV Dsz47 293 Dsz48 262 = Op*46 1 SUM46
290: GTO 258 = xXt R48 - 1 = X 2 = INV Log EE INV EE X xXt = Op*46 Op5
315: Adv Adv Adv 1 INV SUM49 20 S48 2 INV SUM0 .01 S46 50 S47 6 S45
```

```
342:  RO S44 92 B' 95 B' E' Dsz44 352 INV Stflgl xXt 6 x=t 369 1 EE 7
368:  A' 1.376901476 S*47 LB CLR 5 Op17 19 S49 RO S44 R48 S43 Op00
402:  261745 Opl CP 1 INV SUM49 R*49 CP xGEt 434 ± Op2 Op5 R/S Prt S*43
430:  1 INV SUM43 Dsz44 411 Op00 6400 + R20 = Op4 C Adv Op6 Adv Adv
456:  Adv R/S LList Adv Op00 27243637 Op2 Op5 D Adv Adv Adv R/S
```

Note: There are quite a few Dsz register and address operands which must be synthesized.

- - - - - - - - - -

BOOK REVIEW: PROGRAMMABLE CALCULATORS, R J and C J Sippl, 526 pages, Matrix Publishers, 1978.

Charles Sippl (239) and his son Roger cover a lot of ground with this book, and held off publication many months so they could include the latest TI and HP machines. There is a lot of detail covering most (if not all) of the handheld programmables, as well as many of the modern desk-top machines, from hardware descriptions to elementary programming techniques. Considering the scope of this work and time pressures, perhaps the authors may be forgiven for some of the disorganization, repetition, and the errors and lack of clarity in some of the technical text. This work does expose the reader to important material not available elsewhere under one cover, and the serious PPC user will probably find enough helpful information in this book to make it a worthwhile buy, keeping in mind that he may want to consult other sources on important technical topics. And, it is certainly gratifying that Club coverage is both accurate and flattering!

TIPS AND MISCELLANY

More on Strange LRN Behavior (V3N1p5): Lou Cargile (625) reports having experienced a practical problem related to Jared's discovery, and notes 2 situations which can arise during program editing/debugging that can themselves create insidious new bugs: 1) When at step nnn in RUN mode, if you want to get to code headed by say Lbl Tan, it is easy to key GTO Ind by mistake, instead of GTO Tan. Assuming that you want to examine the code, the obvious next step is LRN, which reveals only that you are at step nnn+1, not the desired step, so you key LRN GTO Tan (properly), and proceed on without realizing that a code 22 had been written at step nnn; and 2) If in RUN mode you are single stepping through a sequence of the form: ...Dsz Ind nn N ... and stop after execution of the Ind and key LRN to see where you are, you will see the step containing N, not realizing that the nn in the previous step had been overwritten with code 11. In the first case, if you catch the GTO Ind mistake before pressing LRN, pressing another key (like CLR) first, prevents the unintentional overwrite. In the second case, about all I can suggest is to be wary of SSTing anywhere near an indirect Dsz, and if you stop to see where you are, follow the LRN with BST and verify the displayed code.

Local Club: Dave Johnston (5) and Maurice Swinnen (779) are forming a Washington DC area PPC users group. Contact either Dave or Maurice for more information.

Membership Address Changes: 343: 1325 Quaker St Golden CO 80401; 713: 804 Rhonda Dr Hephzibah, GA 30815; 836: 1539 Rainbow Ln Port Richey, FL 33568; 869: New York, not New Jersey.

Printer Spacing (58/59/PC-100A): Maurice Swinnen (779) has discovered that an Op0 Op5 line-space is 2 mm thinner than one produced by Adv. It turns out that an Op0 Op5 space causes the paper to be advanced exactly as far as occurs following character print, while Adv adds 2 mm. Obvious applications include printer graphics, and paper economy. Lou Cargile (625) found (independently) that Op0 Op5 helped to produce the print format he wanted for his Bridge Deal program (V3N2p5 steps 010-013).

A Tic-Tac-Toe Option (V2N10p6): Dave Leising (890) suggests keying GTO 227 R/S (SBR 227 saves a step) following printing of the first blank grid, if you want the machine to play first.

Basic Hardware Design Information: Dave suggests writing to the U S Patent Office for descriptions of the patents whose numbers are stamped on the back of your machine, as one way to learn more about hardware basics.

Correction (V3N4p1): Prof Kahan is at UC not UCLA.

More on Fractured Display (V2N12p3): Kirk Gregg (748) found that at the time the display fractures, a special state prevails that causes the = or) functions to address Reg 0, i.e. Exc =, STO =, Prd =, etc execute as Exc 0, STO 0, Prd 0, etc. This special state prevails through the use of many built-in functions, but not after keying any of the numerals or CLR. The statistics, D.MS, and P-R functions do not work while this special state prevails. As Kirk notes, this phenomenon doesn't appear to have any useful applications, but a more thorough understanding of it might well lead to important discoveries. Incidently, Jared's fractured digits sequence (V3N1p1) produces the special Reg 0 state, but Fred's (V3N1p6) does not. Neither does Roger's out-of-bounds CROM call method (V3N3p2).

Use of Contributed Material: Maurice Swinnen (779) brings up the matter of member programs or routines submitted to me but which languish unpublished indefinitely, and for which there may be a desire to seek other vehicles for publication. Maurice suggests that a yes or no response from me upon receipt would be helpful, but I'd rather not have to commit myself too soon: topical interest fluctuates over periods of time, and I wouldn't want to reject material that might be just what I'm looking for later on. I suggest that in cases where you wish to publish elsewhere, if your program hasn't yet surfaced in 52-NOTES, send me a clear description of it, and a SASE if you would like it returned. If it is currently in the 52-NOTES publishing cycle, I will so inform you. If it has already appeared in 52-NOTES and you still wish to publish elsewhere, please cite the original 52-NOTES source to help minimize possible copyright contests. For the record: 52-NOTES continues to be published uncopyrighted.

Correction (V3N3p6): Mack Maloney reports that the TI DC 9105 adaptor produces calculator DC, not 120 VAC.

Used SR-52s: Members wishing to sell their SR-52s should contact Harold Bless (255), or a Mr S Green at DAQ Electronics Lackland Dr Middlesex NJ (201) 560-0050.

Merged Code Labels (58/59): Jared Weinberger (221) points out that no mention has been made of the use of the 9 merged codes: 62,63, 64,72,73,74,83,84 and 92 as labels. They all appear to work, but must be created synthetically, like pseudos or double-digit Dsz register addresses, and cannot be addressed from the keyboard.