PPC User Applications Rom
Rom ID 17/18

Application programs from the PPC ROM manual

For the HP 41CL

RomXrom	FatPosition	FatType	FatName	FatGroup	References	Dependencies	Author(s)	Description
17	00	MCODE	-PPC APPS	-PPC APPS	Angel Martin	none	Angel Martin	ppc application programs
17	01	FOCAL	"ACMP"	-PPC APPS	PPC-MAN P.040	none	Roger Hill	indirect mode alpha comparator
17	02	FOCAL	"ACP"	-PPC APPS	PPC-MAN P.099	PR	Ron Yankowski	alpha column print formatting
17	03	FOCAL	"ACV"	-PPC APPS	PPC-MAN P.212	PR	Cliff Carrie	vertical character accumulation
17	04	FOCAL	"AORD"	-PPC APPS	PPC-MAN P.040	none	Roger Hill	indirect mode alphabetizer
17	05	FOCAL	"BLK"	-PPC APPS	PPC-MAN P.320	PPC;PR	Jack Sutton	n/d
17	06	FOCAL	"CAL"	-PPC APPS	PPC-MAN P.320	PPC;PR	Jack Sutton	n/d
17	07	FOCAL	"COMP"	-PPC APPS	PPC-MAN P.028	PPC	Roger Hill	random music composer
17	08	FOCAL	"CPP"	-PPC APPS	PPC-MAN P.100	none	n/a	automatic multiple numeric column formatting
17	09	FOCAL	"CRV"	-PPC APPS	PPC-MAN P.116;PPC-CA V7N5P46	PPC;PR	Bill Barnett	curve fitting program
17	10	FOCAL	"CVPL"	-PPC APPS	PPC-MAN P.116;PPC-CA V7N5P46	PPC;PR	Bill Barnett	curve fitting program
17	11	FOCAL	"FAST"	-PPC APPS	PPC-MAN P.158	PPC	Don Dewey	reducing interest solution time
17	12	FOCAL	"FCT"	-PPC APPS	PPC-MAN P.257	"ISX"	Harry Bertuccelli & Keith Jarett	recursive factorial demonstrating lrs & srs routines
17	13	FOCAL	"HPP"	-PPC APPS	PPC-MAN P.201	PPC;PR	PPC Group Effort	high resolution plot parameters prompting
17	14	FOCAL	"HPT"	-PPC APPS	PPC-MAN P.190	PPC;PR	PPC Group Effort	high resolution plot runtime estimation
17	15	FOCAL	"INIT"	-PPC APPS	PPC-MAN P.257	"ISX"	Harry Bertuccelli & Keith Jarett	recursive factorial initialization
17	16	FOCAL	"IRX"	-PPC APPS	PPC-MAN P.255	PPC	Harry Bertuccelli & Keith Jarett	lrr & srr initialization routine
17	17	FOCAL	"ISX"	-PPC APPS	PPC-MAN P.256	PPC	Harry Bertuccelli & Keith Jarett	lrs & srs initialization routine
17	18	FOCAL	"LBW"	-PPC APPS	PPC-MAN P.027	PPC;WD	Roger Hill	load bytes with wand
17	19	FOCAL	"LPAS"	-PPC APPS	PPC-MAN P.157	PPC;PR	Don Dewey	loan payments and amortization schedule
17	20	FOCAL	"LRR"	-PPC APPS	PPC-MAN P.255	PPC	Harry Bertuccelli & Keith Jarett	lengthen return stack for recursion
17	21	FOCAL	"LRS"	-PPC APPS	PPC-MAN P.256	PPC	Harry Bertuccelli & Keith Jarett	lengthen return stack with single return address
17	22	FOCAL	"MAXMIN"	-PPC APPS	PPC-MAN P.320	PPC	Jack Sutton	print y min & y max foreach 10 values of x between the x limits
17	23	FOCAL	"MIO"	-PPC APPS	PPC-MAN P.265	PPC	John Kennedy	matrix input/output operations
17	24	FOCAL	"MPP"	-PPC APPS	PPC-MAN P.201	PPC;PR	PPC Group Effort	multiple variable plot parameters prompting
17	25	FOCAL	"MPT"	-PPC APPS	PPC-MAN P.190	PPC;PR	PPC Group Effort	multiple variable plot estimation time
17	26	FOCAL	"POP"	-PPC APPS	PPC-MAN P.257	"ISX"	Harry Bertuccelli & Keith Jarett	recursive factorial pop stack
17	27	FOCAL	"PUSH"	-PPC APPS	PPC-MAN P.257	"ISX"	Harry Bertuccelli & Keith Jarett	recursive factorial push stack
17	28	FOCAL	"PHN"	-PPC APPS	PPC-MAN P.347	PPC	n/a	determine phi(n) where n is the absolute value of the intergral part of x
17	29	FOCAL	"RRM"	-PPC APPS	PPC-MAN P.264	PPC	John Kennedy	transform a matrix into a row reduced echelon form
17	30	FOCAL	"SC"	-PPC APPS	PPC-MAN P.439;PPC-CA V7N10P11	PPC	Jake Schwartz & Roger Hill	special printing characters
17	31	FOCAL	"SCDEMO"	-PPC APPS	PPC-MAN P.423	"SC";PR	Jake Schwartz & Roger Hill	sc demo application
17	32	FOCAL	"SHP"	-PPC APPS	PPC-MAN P.201	PPC;PR	PPC Group Effort	high resolution super-plot (multiple paper widths)
17	33	FOCAL	"SMP"	-PPC APPS	PPC-MAN P.201	PPC;PR	PPC Group Effort	multiple variable super-plot (multiple paper widths)
17	34	FOCAL	"SRR"	-PPC APPS	PPC-MAN P.255	PPC	Harry Bertuccelli & Keith Jarett	shorten return stack for recursion
17	35	FOCAL	"SRS"	-PPC APPS	PPC-MAN P.256	PPC	Harry Bertuccelli & Keith Jarett	shorten return stack with single return address
17	36	FOCAL	"SUB1"	-PPC APPS	PPC-MAN P.254	PPC	Harry Bertuccelli & Keith Jarett	demonstration of extended subroutine stack depth
17	37	FOCAL	"SUB2"	-PPC APPS	PPC-MAN P.255	"IRX";"SRR";"LRR"	Harry Bertuccelli & Keith Jarett	lrr & srr demonstration of extended subroutine stack depth
17	38	MCODE	-PPC KAS	-PPC KAS	Angel Martin	none	Angel Martin	key assignments programs
17	39	FOCAL	"C16"	-PPC KAS	PPC-MAN P.294;PPC-TN V1N3P55	none	Bill Wickes	n/d
17	40	FOCAL	"CA"	-PPC KAS	PPC-MAN P.293;PPC-TN V1N3P57	none	Richard Collett	clear assignment
17	41	FOCAL	"CKA"	-PPC KAS	PPC-MAN P.294;PPC-TN V1N3P55	none	Bill Wickes	clear key assignments
17	42	FOCAL	"D^C"	-PPC KAS	PPC-MAN P.294;PPC-TN V1N3P55	none	Bill Wickes	n/d
17	43	FOCAL	"FEA"	-PPC KAS	PPC-MAN P.294;PPC-TN V1N3P55	none	Bill Wickes	find empty assignment

17	44	FOCAL	"GTE"	-PPC KAS	PPC-MAN P.294;PPC-TN V1N3P55	none	Bill Wickes	n/d
17	45	FOCAL	"KA"	-PPC KAS	PPC-MAN P.292;PPC-TN V1N3P57	none	Richard Collett	key assignment
17	46	FOCAL	"MKA"	-PPC KAS	PPC-MAN P.294;PPC-TN V1N3P55	none	Bill Wickes	make key assignment
17	47	FOCAL	"NN"	-PPC KAS	PPC-MAN P.293;PPC-TN V1N3P57	none	Richard Collett	non normalized number maker
17	48	FOCAL	"NNN"	-PPC KAS	PPC-MAN P.294;PPC-TN V1N3P55	none	Bill Wickes	make non normalized number
17	49	FOCAL	"PA"	-PPC KAS	PPC-MAN P.293;PPC-TN V1N3P57	none	Richard Collett	pack assignment
17	50	FOCAL	"PKA"	-PPC KAS	PPC-MAN P.294;PPC-TN V1N3P55	none	Bill Wickes	pack key assignments
17	51	FOCAL	"RAX"	-PPC KAS	PPC-MAN P.294;PPC-TN V1N3P55	none	Bill Wickes	n/d
17	52	FOCAL	"SAX"	-PPC KAS	PPC-MAN P.294;PPC-TN V1N3P55	none	Bill Wickes	n/d
17	53	FOCAL	"VK"	-PPC KAS	PPC-MAN P.447;PPC-CA V7N7P18	none	Richard Collett & Tom Calwallader	view keys assignment
17	54	MCODE	-PPC HANOI	-PPC HANOI	Angel Martin	none	Angel Martin	hanoi tower program
17	55	FOCAL	"GHT"	-PPC HANOI	PPC-MAN P.033	PPC;PR	Harry Bertuccelli	generalized hanoi tower
17	56	FOCAL	"IGT"	-PPC HANOI	PPC-MAN P.033	PPC;PR	Harry Bertuccelli	initialize generalized tower
17	57	FOCAL	"MOVE"	-PPC HANOI	PPC-MAN P.033	PPC;PR	Harry Bertuccelli	move disk from peg
17	58	FOCAL	"PARTS"	-PPC HANOI	PPC-MAN P.033	PPC;PR	Harry Bertuccelli	build up the partitionning
17	59	FOCAL	"SHOW"	-PPC HANOI	PPC-MAN P.033	PPC;PR	Harry Bertuccelli	show current distribution of disks on pegs
18	00	MCODE	-PPC APP2	-PPC APP2	This project	none	Angel Martin	ppc application programs 2
18	01	FOCAL	"ALFA_TN"	-PPC APP2	PPC-MAN P.434	none	n/a	alphabetic tone generator (local labels A to J are used to generate the tone for these letters)
18	02	FOCAL	"K"	-PPC APP2	PPC-MAN P.434	none	n/a	tone for letter K
18	03	FOCAL	"L"	-PPC APP2	PPC-MAN P.434	none	n/a	tone for letter L

-B - STORE PART OF LB

L- and -B together comprise a subroutine version of LB . L- initializes the byte loading process without any prompting, returning to the calling program. -B is then used to load each byte from its decimal equivalent under program control.

Example 1: The following program segment prompts for input and loads an XROM instruction into program memory (after the user has supplied the usual LBL "++" ++...+ XROM LB sequence). This program checks for sufficient SIZE, converts the XROM numbers Y and X to decimal codes for LB , and loads two bytes from the decimal codes. It then prompts for another pair of XROM numbers.

```
01  LBL "XLB"          13  XROM XL
02  12                 14  X<>Y
03  XROM VS            15  STO 05
04  FC?C 25            16  X<>Y
05  PROMPT             17  XROM -B
06  XROM L-            18  RCL 05
07  CF 22              19  XROM -B
08  LBL 00             20  GTO 00
09  "XROM Y, X  ?"     21  LBL 01
10  PROMPT             22  CF 09
11  FC?C 22            23  XROM -B
12  GTO 01             24  END
```

To use "XLB" first key in the LBL "++" ++...+ XROM LB sequence as described in the instructions for LB . Then XEQ "XLB" and supply two XROM numbers in response to the prompt. For instance for XROM 10, 00 key in 10 ENTER↑ 0. Press R/S to calculate and load two bytes. When the next prompt for XROM numbers appears you can either enter another pair of numbers or press R/S without an input to terminate the byte-loading process. The usual prompt "SST, DEL 00p" will be given.

Example 2: If you change line 23 of "XLB" (Example 1) from CF 09 to CF 08, pressing R/S without an input will not terminate the byte loading. Instead, the CF 08 instruction switches to the manual LB operation, allowing additional bytes to be loaded from the keyboard.

COMPLETE INSTRUCTIONS FOR -B

These routines allow bytes to be loaded under the control of your own program . The general rules for their use are as follows:

1. In the program that you are writing which controls the loading of bytes, put the instruction XROM L- . This initializes the byte-loading process and then (instead of prompting for Byte #1) returns to your control program.

2. Have your control program calculate or otherwise place each byte (only decimal allowed in this version) in the X-register, and put an XROM -B in your program to load that byte. Flags 22 and 23 are ignored, as well as whether the calculator is in ALPHA or non-ALPHA mode. The call to routine -B causes one byte to be loaded and then returns to your control program.

3. To terminate the byte-loading process, put the instructions CF 09, XROM -B in your control program. Executing the routine -B with Flag 09 cleared will cause no additional byte to be loaded, but rather a termination of the byte loading, in this case *not* returning to your control program, but ending with a "SST, DEL 00*p*" prompt. Executing -B with Flag 08 cleared will switch from automatic to manual byte loading, allowing more bytes to be loaded directly from the keyboard.

4. Before running your control program, check for size 12, and make room in program memory where you want the bytes to be loaded in exactly the same way as when using the prompting version LB . That is, key in (in PRGM mode) LBL"++", a string of +'s, and XROM LB .

5. Switch out of PRGM mode and instead of pushing R/S to start the byte loader, execute your own control program. Then sit back while your program (if correctly written) calculates, prompts for, or otherwise creates and loads each byte.

6. Execution will terminate with the "SST, DEL 00*p*" prompt, whereupon you can perform the "cleanup" operations just as with the ordinary LB program.

7. If you want your control program to correct a byte that it previously loaded, have it enter a negative number in X and execute -B to get rid of the last-entered byte.

8. Your control program is welcome to make use of any of the contents of registers 06-11 (see above), as long as it doesn't change any of these registers.

WARNING: Don't execute -B (or let your program do it) without having first initialized the process by executing L- ! A few flag and other safeguards have been incorporated, but executing -B by itself *could* cause MEMORY LOST or destruction of existing programs.

When used properly, L- and -B can be very powerful, ultimately allowing one to write a program which writes programs! A somewhat less exotic application is a byte loading program which allows bytes to be scanned in by a wand instead of keyed in.

APPLICATION PROGRAM 1 FOR -B

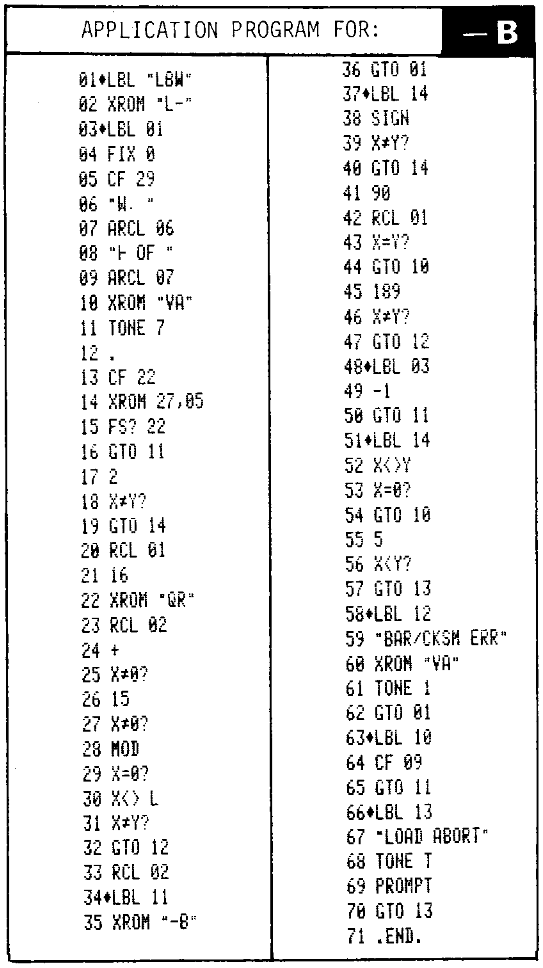
The following program "LBW" (Load Bytes With Wand) allows bytes to be loaded by scanning 2-byte paper keyboard (type 5) barcodes. Only the second byte of the barcode is loaded into program memory, but in order to avoid scanning errors the entire barcode is checked for checksum consistency. Using this program along with a barcode hex table (such as in PPC CJ, V7N6P25-26) and HP's Wand Paper Keyboard, one can rapidly scan in the bytes to be loaded in a manner which for many functions is similar to the normal use of the paper keyboard.

For example, the synthetic instruction X<> can be obtained by scanning X<> in the Paper Keyboard (which will supply the correct prefix) and then byte 78 (hex) in the hex table for the postfix, and TONE 26 can be obtained by scanning TONE from the paper keyboard and byte 26 (decimal) from the hex table. For alpha characters, the barcode hex table can be used, but not the alpha character codes in the Paper Keyboard (or in the character table of PPC CJ, V7N6P23) which use a different format for encoding the character.

Instructions for using "LBW" are as follows:

1. Insert LBL ++, a string of +'s, and XROM LB in the desired part of program memory, just as when using LB .

2. Switch out of program mode and XEQ "LBW". (Note: SIZE 012 or greater is required. If you get the insufficient SIZE message, re-size the calculator



APPLICATION PROGRAM FOR: **—B**

```
01♦LBL "LBW"           36 GTO 01
02 XROM "L-"           37♦LBL 14
03♦LBL 01              38 SIGN
04 FIX 0               39 X≠Y?
05 CF 29               40 GTO 14
06 "W. "               41 90
07 ARCL 06             42 RCL 01
08 "⊢ OF "             43 X=Y?
09 ARCL 07             44 GTO 10
10 XROM "VA"           45 189
11 TONE 7              46 X≠Y?
12 .                   47 GTO 12
13 CF 22               48♦LBL 03
14 XROM 27,05          49 -1
15 FS? 22              50 GTO 11
16 GTO 11              51♦LBL 14
17 2                   52 X<>Y
18 X≠Y?                53 X=0?
19 GTO 14              54 GTO 10
20 RCL 01              55 5
21 16                  56 X<Y?
22 XROM "QR"           57 GTO 13
23 RCL 02              58♦LBL 12
24 +                   59 "BAR/CKSM ERR"
25 X≠0?                60 XROM "VA"
26 15                  61 TONE 1
27 X≠0?                62 GTO 01
28 MOD                 63♦LBL 10
29 X=0?                64 CF 09
30 X<> L               65 GTO 11
31 X≠Y?                66♦LBL 13
32 GTO 12              67 "LOAD ABORT"
33 RCL 02              68 TONE T
34♦LBL 11              69 PROMPT
35 XROM "-B"           70 GTO 13
                       71 .END.
```

NOTE: This program also appears under L- .

and then key in XEQ "LBW" again to restart the process. Just pushing R/S after re-sizing will cause the ordinary **LB** byte loading version to be initiated instead of the wand version.

3. At each prompt "W: N OF M", scan in the appropriate 2-byte barcode (the WNDSCN command is in effect here). After verifying the checksum, the second byte of the barcode will be loaded.

4. A decimal entry can be made directly from the keyboard by clearing the "W: N OF M" prompt (using ←), making the entry, and pushing R/S. Flag 22 is used to detect such an entry. Afer loading the byte, the program will resume with the "W: N+1 OF M" prompt. Hexadecimal entries are not provided for in this program however.

5. To correct an entry, either (a) scan the 1-byte ←barcode, or (b) clear the prompt and XEQ 03, or (c) clear the prompt, enter a negative number, and push R/S. Method (a) can be used to conveniently clear up to 3 bytes by making up to 3 scans at once and waiting while they are processed one by one.

6. During the prompt for a new byte, X=0 while Y= decimal value of previous byte. If you wish to clear the prompt to check the previous byte value, make elementary calculations, etc., push XEQ 01 afterward to get a re-prompt before continuing with the loading.

NOTES

7. To terminate the byte-loading process, either (a) scan the one-byte . (decimal point) barcode, or (b) push R/S twice. Then follow the usual "clean-up" procedures as with **LB** . The loading process will also terminate itself automatically after the maximum number of bytes is reached.

8. If you have accidentally terminated and wish to add more bytes or make corrections, push GTO 03 R/S or GTO 01 R/S (rather than XEQ 03 or XEQ 01, which would disable the return to the "LBW" program).

9. Scanning any 1-byte barcode other than ← or . or any barcode of 3 to 5 bytes will cause the message "BAR/CHKSM ERR" and a re-prompt. The same applies to a 2-byte barcode whose checksum does not check. However, scanning a 6-byte or longer barcode will cause vital information in R06-R11 to be wiped out, so in such a case the whole process is terminated with a "LOAD ABORT" message.

To give a brief analysis of the program:

Lines 01-23 initialize the loading process, and lines 03-14 set up the prompt and execute the WNDSCN command. Lines 15-16 detect an entry from the keyboard and branch to lines 34-35 to load the byte (or backup, if the entry is negative). Otherwise a scan with the wand is assumed to have occured, in which case WNDSCN causes the number of bytes to be in X and the decimal byte values in R01-R0k. If k≠2, a branch is made (lines 17-18) to line 37; otherwise the 4-bit wraparound checksum of the last 3 nybbles is calculated and compared with the first nybble (lines 20-32). A mismatch causes a branch at line 32 to LBL 12 (line 58) where the error message is given; otherwise the second byte of the barcode is recalled and loaded (lines 33-35) and we start over (line 36).

If k≠2 then we had branched to line 37, after which we check for k=1, and if true we check whether the one byte was 90 decimal (5AH, the decimal print code) or 189 (BDH, the back-arrow code) and branch accordingly, otherwise branching to the error message. Lines 51-57 deal with the case where k is neither 1 nor 2; if k=0 then no scan has taken place and it is assumed that R/S R/S was pushed, so we branch to line 63 and initiate the termination procedure by clearing flag 09. If k>5 we branch to line 66 to produce the "LOAD ABORT" message. For other values of k the "BAR/CHKSM ERR" message is produced in lines 58-61, and line 62 branches to a re-prompt.

*The checksum can be calculated by adding up the decimal values of the nybbles; if the result is zero proceed no further. Otherwise take the result mod 15 and if the result of that is zero, change it to 15. In the present case, we are concerned with the last nybble (call it n2) of R01 and both nybbles (call them n3 and n4) of R02, and since n2 + n3 + n4 is equivalent to n2 + 16*n3 + n4 when taken mod 15, it is necessary to decompose the byte in R02 (=16*n3 + n4) into its separate nybbles before adding. Routine QR is used, however, to decompose the byte in R01 into its separate nybbles n1 and n2; n1 is the number to be compared with the calculated checksum.

APPLICATION PROGRAM 2 FOR -B

As an example of a program which writes programs, the following program, "COMP", composes random music by generating a program consisting of tone instructions selected at random from tones 0 through 127 using routine RN to generate the random numbers. To use it, initialize the desired section of program memory with the usual LBL ++, string of +'s, and XROM LB , and then go into non-PRGM mode, make sure the SIZE is at least 012, and execute "COMP". The program will prompt for a seed; enter any number and push R/S. The tone instructions will be loaded into program memory until there is no room left, whereupon the usual "SST, DEL OOP" termination will occur. After performing the usual cleanup operations you can execute your newly composed program and hear the music.

This program can be directly compared with Application Program 2 for TN , "MUS", which generates and plays the tones in "real time". The generation of the random numbers is exactly the same for the two programs (see the description of "MUS" under TN for an explanation), and the tones produced by "MUS" and "COMP" for a given initial seed, will be the same up to the point where the latter runs out of memory space. "MUS" has the advantage of producing tones indefinitely with no initial compilation time, but the listener must put up with the approximately 2-second delay between tones, making the "music" rather tedious. "COMP" requires an initial compilation time (3-4 minutes to generate a 49 tone sequence) and the length of the piece is limited by the number of +'s initially put into program memory, but once the compilation is done the music can be played with no intertone delays. Thus, the results of "COMP" (though they may not become instant hits) are likely to be much more satisfying to the listener.

Lines 02-13 of "COMP" initialize the random numbers (see "MUS" under TN), store frequently used constants, and initialize the byte-loading procedure. Lines 14-21 take the integer part of R07, which is the maximum number of bytes that can be loaded, determine whether it is even or odd, and load a null byte (line 21) if the number is odd. This ensures that there is an even number of bytes left over that can be loaded so we can simply load tone instructions repeatedly (at 2 bytes per instruction) until we run out of bytes, at which time -B will terminate the loading automatically, and the termination will not be in the middle of an instruction. Lines 22-30 form the tone-loading loop in which we first (lines 23-24) load byte 159 (decimal) corresponding to the TONE prefix, then obtain a random number whose integer part is uniformly distributed from 0 to 127 (lines 25-28) and use it for the postfix byte (line 29).

As an aid to the mass production of music (or other byte loading operations) one can record on a single track of a card the following 112-byte program: LBL ++, a string of 104 +'s, XROM LB . (When recording and reading this card there will be a prompt for side 2 which you can ignore and clear). Reading this card and using any of the versions of the byte loader will always allow exactly 98 bytes to be loaded, on our present case allowing 49 tones. The final 49-note piece will then fit onto one track of a card with a few bytes to spare for labels, etc.

As an example of HP-41 generated music, the author found particularly nice the 49-note piece (which coincidentally, takes just 49 seconds to play) obtained by using the card described in the last paragraph and inputting a seed of 4; the initial compilation took 3.25 minutes. If however, the user is not so enthralled by this particular composition, he has plenty of others to choose from. And whether or not he would agree that such music is a manifestation of the true soul of the HP-41, it is undeniable that all of this is an interesting example of calculator composed music, programs that generated programs, and the art of synthetic programming in general. Further refinements could include, for example, weighting factors to favor (say) the short duration tones, and even some "rules of composition" to produce particular musical effects.

BAR CODE ON PAGE 483

APPLICATION PROGRAM FOR: -B

```
01♦LBL "COMP"
02 "SEED?"
03 PROMPT
04 ABS
05 LN
06 ABS
07 FRC
08 STO 00
09 159
10 STO 04
11 128
12 STO 05
13 XROM "L-"
14 RCL 07
15 INT
16 2
17 MOD
18 1
19 -
20 X=0?
21 XROM "-B"
22♦LBL 01
23 RCL 04
24 XROM "-B"
25 CLX
26 XROM "RN"
27 RCL 05
28 *
29 XROM "-B"
30 GTO 01
31 .END.
```

LINE BY LINE ANALYSIS OF -B

See LB .

CONTRIBUTORS HISTORY FOR -B

L- and -B were conceived and written by Roger Hill (4940) as an integral part of the ROM version of LB .

AL - ALPHABETIZE X & Y

AL is a general-purpose alphabetizing subroutine. It compares two alpha strings and, if they are not already in proper alphabetical order, exchanges them.

AL may be used in either of two modes, which are selected automatically according to the nature of the contents of the X & Y registers: (1) Direct mode- If the X & Y registers contain alpha strings, XEQ AL will leave them in ascending alphabetical order in X & Y; that is, the string which is "lower" or closer to the beginning of the alphabet will be left in X, and the "higher" string in Y. (2) Indirect mode- If the X & Y registers contain numbers, AL will interpret them as indirect addresses and will alphabetize the character strings in the two data registers addressed by X & Y. If the register addresses in X & Y are in ascending order, the strings will be alphabetized in ascending order; if the register addresses in X & Y are in descending order, the strings will be alphabetized in descending order.

Example 1: Direct mode. (d= don't care, g= garbage)

T	d		T	g
Z	d		Z	g
Y	"ALPHA"	XEQ AL →	Y	"BETA"
X	"BETA"		X	"ALPHA"
L	d		L	g
ALPHA	d		ALPHA	[clear]

T	d		T	g
Z	d		Z	g
Y	"BETA"	XEQ AL →	Y	"BETA"
X	"ALPHA"		X	"ALPHA"
L	d		L	g
ALPHA	d		ALPHA	[clear]

Example 2: Indirect mode (ascending).

T	d		T	90
Z	d		Z	89
Y	90	XEQ AL →	Y	g
X	89		X	g
L	d		L	g
ALPHA	d		ALPHA	[clear]
R89	"BETA"		R89	"ALPHA"
R90	"ALPHA"		R90	"BETA"

Example 3: Indirect mode (descending).

T	d		T	89
Z	d		Z	90
Y	89	XEQ AL →	Y	g
X	90		X	g
L	d		L	g
ALPHA	d		ALPHA	[clear]
R89	"ALPHA"		R89	"BETA"
R90	"BETA"		R90	"ALPHA"

COMPLETE INSTRUCTIONS FOR AL

The alpha strings on which AL operates may be of different lengths, up to the maximum of six characters which can be held in a data register as a result of the ASTO command. For example, "AAAA" will be placed ahead of "AAAAAA". Any character in the 41C's character set (including those from the lower half of the combined hex table) can be included, and they will be "alphabetized" in the order of their decimal or hex numbers as set forth in the combined hex table. For example, "3BB" will be placed ahead of "4AA". In terms of printable characters, the strings will be alphabetized in the order of their "BLDSPEC" numbers. One unfortunate consequence to remember (common to all systems using ASCII codes) is that the entire set of lower case letters comes after the entire set of upper case letters, so that "alpha" will be placed after "BETA".

Stack usage is shown in the Examples, above.

APPLICATION PROGRAM 1 FOR AL

The routine ACMP listed below is a faster alphabetizer that works only in the indirect mode. In that mode it operates identically to AL . The AORD routine below accepts an ISG/DSE pointer of the form bbb.eeeii and uses ACMP to alpha-sort the contents of the chosen block of registers in ascending order. Both routines were written by Roger Hill (4940). If you need more speed than AL and you don't need the direct mode, or if you want to sort alpha data, use these routines.

BAR CODE ON PAGE 484

APPLICATION PROGRAM FOR: AL

```
01♦LBL "AORD"      24 ISG Y          45 X<> IND Z
 02 CF 10          25 GTO 05         46 X<> IND T
 03 ENTER↑         26 RTN            47 X<> IND Z
 04 ENTER↑                            48 RTN
 05 FRC          27♦LBL "ACMP"       49♦LBL 03
 06 ST- Y           28 SF 09          50 R↑
 07  E-3                              51 R↑
 08 ST- T          29♦LBL 01         52 SF 09
 09 ST* Z          30 "*x"
 10 /            31 ARCL IND Y       53♦LBL 04
 11 +              32 "⊦♦♦♦♦"        54 "**"
                   33 ASTO [       55 ARCL IND Y
 12♦LBL 05         34 "⊦♦♦"          56 ASHF
 13 ABS            35 RCL [          57 ASTO T
 14 X<>Y           36 FS?C 09        58 "x"
                   37 GTO 01         59 ARCL T
 15♦LBL 06         38 X<Y?           60 "⊦♦♦♦♦♦"
16 XEQ "ACMP"      39 RTN            61 ASTO [
 17 R↑             40 X=Y?           62 "⊦♦"
 18 R↑             41 GTO 03         63 RCL [
 19 DSE Y                            64 FS?C 09
 20 GTO 06        42♦LBL 02          65 GTO 04
 21 LASTX          43 FS?C 10        66 X>Y?
 22  E-3           44 RTN            67 GTO 02
 23 +                                68 .END.
```

LINE BY LINE ANALYSIS OF AL

The byte manipulation undertaken by the routine at LBL 14 left-justifies shorter strings in the register, else leading null bytes would result in shorter strings being alphabetized ahead of longer strings even if their first character was high^r. The left-justification is cleverly achieved by lines 158-160. The alpha register was first cleared to nulls at line 121. Line 158 then pushes even a one-character string into the sixth position from the right, by appending five nulls. ASTO L then stores six alpha bytes into the L register, beginning with the left-most character in the alpha register -- that is, with the first character of the original string. In the case of a one-character string, for example, ASTO L places that one character followed by the five appended null bytes into the L register. Finally, ARCL L appends the resulting six-character string back onto the right end of the alpha register, whence it is pushed five places to the left by line 161.

The second LBL 14 at line 167 then obliterates the trailing two bytes of the string by storing 0 in the M register, pushes the string the rest of the way into N, and recalls it into X. At this point, the first four

1304.5	1,304.5	3rd value into
XEQ CP	1,304.5	print buffer
ENG 0	1. 03	4th display mode
CF29	1. 03	and commas off
3 STO 06	3. 00	4th skip index
3 EEX 6 CHS	3. -06	4th value into
XEQ CP	3. -06	print buffer
PRBUF	3. -06	Prints last line

The above keystroke sequence would print the first line of table 1. To print the other lines of the table, one would follow the sequence for the first line, except to substitute the line 2 values 3.26, 8, 6814.3 and 1 EEX +30 for line 1 values 3.21, 8, 1304.5 and 3 EEX -6, then line 3 values, and finally the last set of values.

```
  3.21  2 1,304.5 3.-06
 43.26  8 6,814.3 1.+30
  0.58 10 1,313.1 6.-09
618.18  1 4,441.6 3.-12
```

MORE EXAMPLES OF CP

Example 2. Print the information in table 3 on the 82143A printer using the CP routine:

ROM PERIPHERAL ROUTINES:

NAME	BYTES	DEVICE	SIZE
LG	45	PRINTER	0
HS	40	PRINTER	6
HA	50	PRINTER	6
CP	60	PRINTER	7
BA	337	WAND	19
MP / HP	596	PRINTER	35

Table 3. Information to be printed using the CP routine for example 2

Here is a case where we must have both ALPHA and numeric columns in the same printed lines. The length of the ALPHA information is not consistent down the two ALPHA columns, so there should be a way that the 41C can know how to left justify the ALPHA entries. Below is a routine, written by Ron Yankowski (2980) which left justifies ALPHA entries.

ALPHA Column Print Formatting: This routine will left-justify data in the ALPHA register and accumulate it into the print buffer. If the information is shorter than a user designated length, then spaces will be added to fill the remaining columns. If the ALPHA is too long, the string will be truncated at the designated length. The width may be from 1 to 18 characters. The instructions are listed below:

Keystrokes	Display	Result
N	N	Enter maximum column
STO 07	N	width (18 or less)
ALPHA (text) ALPHA	(text)	Key the text into the ALPHA register
XEQ ACP	(text)	Text is added to the print buffer left justified

The column width value in register 07 remains unchanged after executing ACP, so it does not need to be reloaded if the same column is being left justified repeatedly. The listing of ACP is below:

BAR CODE ON PAGE 479

APPLICATION PROGRAM FOR: CP

```
01♦LBL "ACP"
02 6
03 RCL 07
04 "⊦      "
05 X<=Y?            1 to 6 char's long
06 GTO 14
07 RCL Y
08 -
09 ASTO Z
10 ASHF
11 SF 10
12 "⊦      "
13 X<=Y?            7 to 12 char's long
14 GTO 14
15 RCL Y
16 -
17 ASTO T
18 ASHF
19 SF 09
20 "⊦      "        Greater than 12
21♦LBL 14           characters long
22 -
23 ASTO T
24 CLA
25 X>0?
26 XEQ 13
27 ARCL T
28 ASTO X
29 LASTX
30 CLA
31 XEQ 13
32 ARCL Y           Restoring ALPHA
33 ASHF             register
34 ASTO X
35 CLA
36 FS?C 10
37 ARCL Z
38 FS?C 09
39 ARCL T
40 ARCL X
41 ACA
42 RTN
43♦LBL 13           Append X no. of
44 "⊦ "             blanks onto string
45 DSE X
46 GTO 13
47 END
```

Routine ACP uses R07 and flags 10 and 09 as well as the stack. It leaves ALPHA intact for later use.

Returning to example 2, we nay now use the ACP routine to create both of the ALPHA columns in the example. Use the column width values of 5 and 7 for ALPHA columns 1 and 2 respectively. The first numeric column is FIX 0 with no commas and 3 maximum digits (skip index = 2 from table 2), and the second numeric column is FIX 0 with 2 maximum digits left of the decimal point. However this second numeric column is an extra 2 characters to the right of the previous one, allowing a position for the sign. Therefore, use 2+2 or 4 digits, yielding a skip index of 3 from table 2. The resulting keystroke sequence is:

KEYSTROKES	DISPLAY	RESULT
ALPHA ROM (space) PERIPHERAL (space ROUTINES: ALPHA	(text)	Enter header
XEQ PRA	(text)	Print line
XEQ ADV		Skip a line
ALPHA (space) NAME (space) BYTES (space SIZE (space) DEVICE ALPHA	(text)	Header
XEQ PRA	(text)	Print header
FIX 0		Set display mode
5 STO 07	5	1st ALPHA column
ALPHA LG ALPHA	LG	ALPHA entry
XEQ ACP	LG	Left justifies
2 STO 06	2	1st skip index
45 XEQ CP	45	Add to buffer
1 SKPCHR	1	Skip a space
7 STO 07	7	2nd ALPHA column
ALPHA PRINTER ALPHA	(text)	ALPHA entry
XEQ ACP	(text)	Left justifies
4 STO 06	4	2nd skip index
0 XEQ CP	0	Add to buffer
XEQ PRBUF	(text)	Prints buffer
5 STO 07	5	1st ALPHA column
ALPHA HS ALPHA	HS	ALPHA entry
XEQ ACP	HS	Left justifies
2 STO 06	2	1st skip index
40 XEQ CP	40	Add to buffer
7 STO 07	7	2nd ALPHA column
ALPHA PRINTER ALPHA	(text)	ALPHA entry
XEQ ACP	(text)	Left justifies
4 STO 06	4	2nd skip index
6 XEQ CP	6	Add to buffer
XEQ PRBUF	6	Prints buffer

etc.
(Continues for lines 3 to 6 similarly.)

The Printer Preparation Form.

In order to better prepare printer outputs for column alignment, a form has been provided which allows composition of the full 24-character lines for determination of CP skip indexes. Along with the printer columns, the format of each column may be included, for easier programming. Remember that for columns which will be aligned by CP, an extra space must be allotted for a sign position, whether one is present or not. This is because CP uses function ACX, which leaves room for the sign before the number. Since one would usually leave a space between columns anyway, this is not a problem. However, if an extra space is inserted, then 2 spaces will appear if all the numbers in the column are positive.

Two copies of the preparation form are included. The first is filled out for the two previous examples. The other is blank, and should be photocopied for use in preparing future outputs requiring CP .

Automatic Multiple Numeric Column Formatting:

Routine CPP is one which automates the formatting of multiple columns of all-numeric information. It can also be used before or after ALPHA columns have been placed in the buffer, leaving a string of consecutive numeric columns to be added. The instructions are shown below:

Load the data registers with the information required for the first line of the table:

R08 = bbb.eee where bbb=first reg. of data
eee=last reg. of data

R(bbb) = 1st column numeric value

R(bbb+1) = ±a.bc where a = skip index for 1st column
b: 1=FIX, 2=SCI, 3=ENG
c = # display digits (0 to 9)
+a.bc = CF29 (no commas), -a.bc = SF29 (commas)

R(bbb+2) = 2nd column value

R(bbb+3) = ±a.bc for second column

. .

. .

R(eee) = nth column value

R(eee+1) = ±a.bc for nth column

Place any ALPHA information in the buffer, then XEQ CPP. Now add trailing ALPHA if any, and PRBUF and the line is printed. The procedure for each successive line of the printed table is: Accumulate columns into the buffer, load data registers, XEQ CPP for the string of consecutive numeric columns, add any other columns to the buffer, then PRBUF. The listing of CPP is below:

BAR CODE ON PAGE 479

APPLICATION PROGRAM FOR: **CP**

```
01•LBL "CPP"
02 CF 29
03 2 E-5
04 ST+ 08          Set counter in R08
05•LBL 00
06 RCL 08
07 1
08 +
09 RCL IND X       Recall next register
10 X<0?
11 SF 29           Test for commas
12 ENTER↑
13 INT
14 ABS
15 STO 06          Store skip index
16 RDN
17 FRC
18 10
19 *
20 ENTER↑
21 INT
22 X<>Y
23 FRC
24 10
25 *
26 X<>Y
27 1
28 X=Y?
29 FIX IND Z       Testing for
30 RDN             specified
31 2               display mode
32 X=Y?
33 SCI IND Z
34 RDN
```

display should show 6 after entering the first new data pair below.

Do:	See:
4 CHS ENTER↑ 0.713 [Σ+]	6.0000
2.5 ENTER↑ 10.93 [Σ+]	7.0000
6 ENTER↑ 47.53 [Σ+]	8.0000
10 ENTER↑ 254.95 [Σ+]	9.0000

For a new linear fit key 1 [SOLVE TYPE j]. The data returned is:

```
Z: 0.765698771 = r
Y: 0.978958100 = a
X: 15.33154618 = b
```

For a new exponential fit key 2 [SOLVE TYPE j]. The data returned is:

```
Z: 0.993615263 = r
Y: 3.825595338 = a
X: 0.419945301 = b
```

Now choosing the best r we see that the new data reflects a change in the curve type. Since the exponential parameters should still be in the^machine we can predict y when x = -10. Key 10 CHS [ŷ]. y = 0.057398396.

Example 4: Fit the best curve to the following set of data points.
(1, 2), (2, 2.828), (3, 3.464), (4, 4), (5, 4.472), (6, 4.899), (7, 5.292), (8, 5.657), (9, 6).

In this example the x-coordinates start counting from 1 and are consecutive integers. So we need only input the y-coordinates, but they must be in the proper order. The count in the display will serve as the x-coordinates.

Do:	See:
[INITIALIZE]	1.0000
2 [Σ+]	2.0000
2.828 [Σ+]	3.0000
3.464 [Σ+]	4.0000
4 [Σ+]	5.0000
4.472 [Σ+]	6.0000
4.899 [Σ+]	7.0000
5.292 [Σ+]	8.0000
5.657 [Σ+]	9.0000
6 [Σ+]	10.0000

Since all the data are positive we may use the best fit function to let the program find the best fit among all 4 curve types. Press [SOLVE BEST]. The contents of the stack when the program stops are:

```
T: 0.999999994 = r
Z: 1.999855865 = a
Y: 0.500043886 = b
X: 4.000000000 = best curve type
```

This indicates a power curve (type 4) where the equation is of the form:

$y = (2.00)*x^{0.50}$ (values rounded to 2 places)

APPLICATION PROGRAM 1 FOR CV

Curve fit solutions are often more meaningful when the points input are also plotted, superimposed on the plot of the "best fit" or selected equation type. The CVPL program will function exactly as CV functions and, after calculating the parameters a, b, r and r↑2, the program will stop with the prompt: "TO PLOT: R/S" To plot the equation calculated with the points input superimposed, simply press the R/S key. Nothing else need be done to obtain a plot. When accomplished in this way, the default situation, all numbers will be printed with 2 decimal places and the resulting plot will contain 50 plotted points. The detailed instructions include options to print other than 2 decimal places and to plot a smaller or greater number of points. The same key captions used by CV are used, plus the shifted keys b, c, and d for the optional features indicated.

Note that this program can also be used without the printer and will function essentially the same as CV but with the display labeling the points entered, showing deletions indentified as such, and labeling the parameters calculated.

The plotting program takes into account all possibilities: duplicate, identical points; almost identical points that would plot as identical; points with identical x-values but with significantly different y-values; individual single points. Any quantity of duplicate points can be handled. The points are plotted with 4 plotting characters as follows:

a. The equation of type J is plotted using a small square dot (box). One equation point is normally plotted before the first input point and after the last point. If the first input point is close to zero, it will be plotted first.

b. Individual single points are plotted with a large X.

c. Two (or more) essentially identical points are plotted with a double X, two small x's, one above the other.

d. Two (or more) points having essentially the same x-value but having different y-values are plotted with an asterisk located where the largest of the point's (based on x-value) plot should be. If desired the other points not shown for this value of x could be drawn in by hand or more points could be selected for the plot to separate very close x-values.

To simplify the program and reduce the number of registers required to store the data points, both the x- and y-value of a point are stored in one register, using a decimal point to separate them. This limits the magnitude and sign of the numbers to the following: data points must be nonzero, positive numbers and less than 1000 in magnitude. If you need to deal with larger numbers, shift all decimal points before entering them. Note: if the program is used without the printer, or by pressing "NO PLOT" with a printer, none of these restrictions apply and the "data error" message will not be encountered if you try to use negative or large numbers. See the valid use of negative entries in the CV instructions, however.

This program was developed originally as a modification to Gary Tenzer's curve fit program, "CFIT" in the PPC JOURNAL, V7N5P46, and was to be published in the JOURNAL as a stand alone program. The program was 691 steps in length (1414) bytes. With the CV routine (plus others such as the S2 sorting routine) the plotting routine was completely re-written to utilize as many of the ROM routines as possible and the end result is presented below, significantly improved over my earlier version, with 439 steps and using 865 bytes (8 tracks on 4 cards). Most of Gary's displays and labeling are used in this program which partially account for the length of the program. I feel these extras are desirable, especially when using a printer.

Example 1 for CVPL: Use the same problem as Example 1 for CV. Find the linear equation for the following data: (1.1, 5.2), (4.5, 12.6), (8.0, 20.0), (10.0, 23.0), (15.6, 34.0).
Then predict y when x=20 and predict x when y=25.

Plug the PPC ROM in and using the card reader, read in all 8 sides of the program CVPL. Put the calculator in USER mode. Connect printer and put in MAN mode. Press Initialize (shift E) and the display will tell you to SIZE 038 plus the number of points you plan to input. For this example SIZE 043 (=38 + 5 points). Press R/S to complete intialization of the program. See 1.00 in the display asking for the first point's values. First however, we will select 4 decimal places in the printout so key in 4 and press shift C (for the number of decimal places). See 1.00 again asking for the first point's values. Key in each point exactly as in the CV instructions by keying in X ENTER↑ Y [Σ+]. Keyboard functions assigned to keys are shown in square braces [ ] below.

Do:	See:
[Initialize]	"SIZE=38+ PTS"
XEQ "SIZE" 043	0.00 (size=38+5)
R/S	1.00 TONE 9
4 [No.Dec.Places]	1.00 TONE 9
1.1 ENTER↑ 5.2 [Σ+]	X1=1.0000, Y1=5.2000 2.0000 TONE 9
4.5 ENTER↑ 12.6 [Σ+]	X2=4.5000, Y2=12.6000 3.0000 TONE 9
8.0 ENTER↑ 20.0 [Σ+]	X3=8.0000, Y3=20.0000 4.0000 TONE 9
10.0 ENTER↑ 23.0 [Σ+]	X4=10.0000, Y4=23.0000 5.0000 TONE 9
15.6 ENTER↑ 34.0 [Σ+]	X5=15.6000, Y5=34.0000 6.0000 TONE 9

Since we want a linear curve, we key in 1 and push [SOLVE TYPE J]. When execution stops the following will be printed.

```
1: LIN
a=3.4991
b=1.9720
r=0.9990
r↑2=0.9981
```

In the calculator display see "TO PLOT: R/S"
If we now press R/S the plot will consist of 50 points. To select plot of 25 points, key in 25 and press [No.Pts.In Plot], the shifted B key. The same display will appear in the calculator (nothing is printed). Before plotting, we will first find the predicted y and x values asked for. Key in 20 and push [$\hat{y}$], the C key. Printed (and displayed) see:

"IF X = 20.0000, Y = 42.9401"

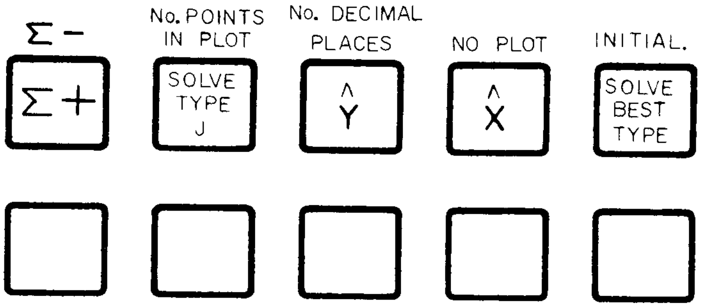
Key in 25 and push [$\hat{x}$], the D key and see:

"IF Y = 25.0000, X = 10.9028"

Now press R/S to plot the data. When the plotting is complete, wait for the BEEP before stopping the calculator.

The total time for this example, except for sizing the calculator was 4 min. and 25 sec. The primary consumer of time is normally the plotting, so the number of points selected greatly effects execution time. Often a short plot of 15 points is adequate.

After the BEEP has sounded the completion of the plot you can find other predicted values of x or y, select a different curve type, add points or delete points and see the effect on the new plot.



INSTRUCTIONS	INPUT DATA	KEYS	OUTPUT
1. Load cards,sides 1-8 in USER mode			0.00
2. Initialize		shift E	"SIZE=38+PTS"
3. If SIZE inadequate Otherwise go to step 4		XEQ "SIZE"	
4. Complete initialization		R/S	TONE 9 1.00
5. Optional - To print without plotting (including negative or larger numbers)		shift D	1.00
(Note: for new problem with plotting, must CF 24)			
6. Optional - Select no. of decimal places to be printed. Default is 2. Or key in n	n	shift C	TONE 9 1.00

7. Enter data point	X
	ENTER↑
	Y
		[Σ+]	Xj=--.----
			Yj=--.----
			TONE 9
			# next point

Note: where x-values are same as displayed # of next point, input only Y and press A			
8. If point input is correct go to step 9. If incorrect, press R/S to delete the point just entered.		R/S	"**DELETE**" X=--.---- Y=--.---- TONE 9 # next point
To delete any previously entered point, re-enter exact X & Y values and press			

[Σ-] (same)

9. For each new point wait for TONE 9 and repeat step 7. (same as 7)

Note: Program will accept only positive values of X and Y in the range .01-999.99. For numbers outside of range shift decimal before entering. For a zero value use .01. "DATA ERROR" message will be displayed after an invalid entry. This note only applies with printer connected. Any values for X and Y will be accepted without a printer or after pressing "NO PLOT" with a printer. See CV instructions regarding acceptable negative numbers.

10. Calculate a,b,r,r^2:

a. For "best fit" based on largest ABS value of r: E (typical)
"1:LIN"
"a=--.----"
"b=--.----"
"r=--.----"
"r↑2=--.----"
"TO PLOT: R/S"

(Note: r & r^2 display correctly only on printer. Final caption not shown if printer not connected)

b. For selected type "j" curve Case:

1: Linear	1 B	
2: Exponential	2 B	(same)
3: Logarithmic	3 B	
4: Power	4 B	

NOTE: Step 10 must be accomplished after all data points have been entered before steps 11, 12, or 13 may be attempted.

11. Optional: select number of points to be plotted (points input plus equation points).
a. Default value = 50 points no action required.
b. Enter # of desired points

n shift B "TO PLOT: R/S"

12. Project y given x $\hat{x}$ C "IF X = --.----"
"Y = --.----"
"TO PLOT: R/S"

13. Project x given y $\hat{y}$ D "IF Y = --.----"
"X = --.----"
"TO PLOT: R/S"

14. To add additional points to same data, go to step 7.

15. Plot curve and data points R/S Curve and points plotted

The following symbols are used:

▪ points on curve type "j"
× data points, no duplicate X or Y value
⨳ 2 or more data points with the same X and Y values within the plotting tolerance.
✳ 2 or more data points with same X-value but different Y-values. Only one of the points is plotted.

BEEP sounds after plot is complete

Note: after plotting wait for BEEP. Then you can add more points, delete points, predict new X or Y values, plot with a different number of points, calculate curve parameters with a different number of decimal places displayed or select a different curve type by going back to the above instructions.

Example 2 for CVPL: This example will demonstrate all four plotting characters described above and show how deletions and points can be added. The initial points are the following:

(70.00, 11.10), (10.40, 71.86), (22.30, 38.71),
(10.50, 73.12), (40.90, 21.73), (4.20, 85.20)
(100.30, 1.34), (41.30, 34.70)

Print with 4 decimal places and solve for the best fit curve. Then find the predicted value of y for X=35 and the predicted value of X for Y=100. Then plot using 30 points in the plot. Size for one additional point to be added. In the following the data in parentheses are not printed.

Do:	See:
[initialize]	("SIZE=38+PTS")
XEQ "SIZE" 047	(0.00)
R/S to complete initialization	(1.00 TONE 9)
4 [# dec. places]	(1.00 TONE 9)
70 ENTER↑ 11.1 [Σ+]	"X1=70.0000" "Y1=11.1000" (2.0000 TONE 9)
10.4 ENTER↑ 71.86 [Σ+]	"X2=10.4000" "Y2=71.8600" (3.0000 TONE 9)
22.3 ENTER↑ 38.71 [Σ+]	"X3=22.3000" "Y3=38.7100" (4.0000 TONE 9)
10.5 ENTER↑ 73.12 [Σ+]	"X4=10.5000" "Y4=73.1200" (5.0000 TONE 9)
40.9 ENTER↑ 21.63 [Σ+]	"X5=40.9000" "Y5=21.6300" (6.0000 TONE 9)

Y5 was entered in ERROR so to delete:

R/S	"DELETE" "X=40.9000" "Y=21.6300" (5.0000 TONE 9)

Now continue entering the correct values

40.9 ENTER↑ 21.73 [Σ+]	"X5=40.9000" "Y5=21.7300" (6.0000 TONE 9)
4.2 ENTER↑ 85.2 [Σ+]	"X6=4.2000" "Y6=85.2000" (7.0000 TONE 9)
100.3 ENTER↑ 1.34 [Σ+]	"X7=100.3000" "Y7=1.3400" (8.0000 TONE 9)
41.3 ENTER↑ 34.7 [Σ+]	"X8=41.3000" "Y8=34.7000" (9.0000 TONE 9)
Now push E for [SOLVE BEST]	"3: LOG" "a=132.4456" "b=-28.2822" "r=-0.9812" "r↑2=0.9627" ("TO PLOT: R/S")

Find the predicted values:

```
35 [ ŷ ]                      "IF X=35.0000"
                            "Y=31.8925"
100 [ x̂ ]                     ("TO PLOT: R/S")
                            "IF Y=100.0000"
                            "X=3.1494"
                            ("TO PLOT: R/S")
```

Now select a 30 point plot:

```
30 [# points in plot]    ("TO PLOT: R/S")
R/S to plot the data
```

After the BEEP sounds and the plotting is complete, add an additional point (71.1, 11.0), almost the same as point 1, and delete what appears to be the worst fitting point (22.30, 38.71).

```
71.1 ENTER↑ 11.0  [Σ+]  "X9=71.1000"
                          "Y9=11.0000"
                          (10.0000 TONE 9)

22.30 ENTER↑ 38.71  [Σ-]
                          ** DELETE **
                          "X=22.3000"
                          "Y=38.7100"
                          (10.0000 TONE 9)
```

Now again solve for the best fit.

```
[SOLVE BEST]              "3: LOG"
                          "a=133.8645"
                          "b=-28.5171"
                          "r=-0.9858"
                          "r↑2=0.9719"
                          ("TO PLOT: R/S")
```

We have slightly improved the fit to a log curve and the parameters a and b have of course changed. Now make a new plot by pressing R/S. After replotting the data, again find the predicted values of y if x=35 and x if y=100.

```
35 [ ŷ ]                        "IF X=35.0000"
                              "Y=32.4761"
100 [ x̂ ]                       "IF Y=100.0000"
                              "X=3.2789"
```

Looking at the plot, note the value of having the first input point be preceded by a point on the LOG curve. Note the double x at x=11 representing 2 almost identical points X2 and X4. The asterisk at x=41 means 2 or more points have essentially the same x-value but very different y-values. They are X5 and X8 and because X8 has a larger x-value than X5, the asterisk is plotted for Y8.

LINE BY LINE ANALYSIS OF CVPL

Lines 02-11 set up default conditions for 50 point plot and 2 decimal place printout. Lines 14-21 display next point to be input. Lines 22-30 are the delete routine using R/S. Lines 31-70 are the delete routine for later deletion of a point which first combines x and y in a single number as YYYYY.XXXXX after rounding to 2 decimal places, then searches stored points registers for the same point. When the point is found a copy of the last point stored is made in that register. Flag F05 prevents display of point number for a delete. Input of new points are added to CV statistical registers (71-118), then x and y values are checked for sign and magnitude and rounded to 2 decimal places and stored in YYYYY.XXXXX format. Lines 86-186 recall full numbers (not rounded) from CV for printing to number of decimal places selected and printout is formatted for input points, deleted x and y, and calculated parameters a, b, r, and r↑2. Lines 187-192 display plotting prompt "TO PLOT: R/S" only if printer connected, so program can be used without printer. Lines 193-200 store the barcoded input plotting symbols. Lines 201-217 exchange registers R07-11 with R33-37 using BE so data needed for CV statistical registers will be saved for later use, not lost when "PRPLOT" in printer ROM uses registers R07-11. Lines 218-236 use BX to find maximum and minimum y values of input points, then increase maximum and decrease minimum y by 25% of range to allow for equation points to be plotted outside of range of input points. Lines 237-241 make Ymin=0 if this value would have become negative after the 25% adjustment. These lines also determine the y-plotting increment used to see if 2 points have essentially same y-value. Lines 244-258 store "CRV" as the curve name for PRPLOT. The next function performed is a reverse of the left and right sides of the decimal point. Points are now stored as XXXXX.YYYYY (244) and S2 is used to sort the stored points to find maximum and minimum x and for faster plotting (246). Also calculated is the x-plotting increment using the range of x-values and number of points wanted in plot. If the x-minimum is smaller than plotting increment, lines 259-266 make the 1st point plotted the smallest x-value of the points; otherwise the x-minimum is set so one equation point will be plotted first. X-max made large enough that PRPLOT will never stop plot so one equation point can be plotted after largest x-values of input points (267-275). Stop routine initiated when one equation point beyond last point has been plotted. Lines 277-292 restore the statistical registers for CV by XEQ BE , then reverse stored points to original YYYYY.XXXXX format (284). Lines 293-296 reset the counter and "BEEP", ready for changes to data, etc. Flags 02 and 00 are used to determine if plotting is complete, lines 329-330. Routine to check stored points to see if they should be plotted at this x-value (297-323), checks +50% of plotting increment from this plotting point. If flag F03 is set (324) at least one point to be plotted here, and still checking for others. Plotting symbol to be used selected (340-360) and stored in R03 for "PRPLOT" to use for plotting. Where 2 input points have essentially the same x-value, checks to see if their y-values are also essentially the same (361-378). Flag F04 is set when 2 points have the same y-values, F01 is set when they have significantly different y-values (375-377). Plotting routines for the 4 curve types are in steps 379-399. The routine to reverse the left and right sides of the stored points (from the decimal point) is LBL 16, steps 400-419. Storage routines for optional selection of number of points in plot and number of decimal places in printout are in steps 425-435.

NOTE: The BLSPEC numbers for the plotting characters, if barcodes are not used, are:

box:	0,	0,	28,	28,	28,	0,	0
large X:	0,	34,	20,	8,	20,	34,	0
double x:	0,	0,	73,	54,	54,	73,	0
asterisk:	0,	20,	8,	62,	8,	20,	0

The ROM routine BL can also be used to create the equivalent of these BLDSPEC characters.

BAR CODE ON PAGE 481

APPLICATION PROGRAM FOR: **CV**

```
01♦LBL "CVPL"
02♦LBL e
03 4900
04 STO 29
05 2
06 STO 38
07 .
08 "SIZE=38+ PTS"
09 PROMPT
10 STO 06
11 XROM "CV"
12 39.999
13 STO 30
14♦LBL 12
15 RCL 18
16 1
17 +
18 CLA
19 ARCL X
20 TONE 9
21 PROMPT
22 DSE 30
23 SIN
24 SF 10
25 6
26 STO 06
27 RCL 08
28 RCL 09
29 XROM "CV"
30 GTO 08
31♦LBL a
32 SF 10
33 6
34 STO 06
35 RDN
36 XROM "CV"
37 RCL 08
38 RND
39 1 E3
40 /
41 STO 00
42 RCL 09
43 RND
44 1 E2
45 *
46 ST+ 00
47 RCL 30
48 1
49 -
50 STO 27
51 39.999
52 STO 30
53♦LBL 13
54 RCL IND 30
55 RCL 00
56 X=Y?
57 GTO 11
58 ISG 30
59 GTO 13
60♦LBL 11
61 RCL IND 27
62 STO IND 30
63 RCL 27
64 STO 30
65♦LBL 08
66 SF 12
67 "** DELETE **"
68 AVIEW
69 SF 05
70 GTO 09
71♦LBL A
72 1
73 STO 06
74 RDN
75 XROM "CV"
76 RCL 08
77 XEQ 14
78 1 E3
79 /
80 STO IND 30
81 RCL 09
82 XEQ 14
83 1 E2
84 *
85 ST+ IND 30
86♦LBL 09
87 SF 12
88 FIX 0
89 "X"
90 FC? 05
91 ARCL 18
92 FIX IND 38
93 "⊢="
94 ARCL 08
95 AVIEW
96 PSE
97 FIX 0
98 "Y"
99 FC? 05
100 ARCL 18
101 FIX IND 38
102 "⊢="
103 ARCL 09
104 AVIEW
105 PSE
106 ADV
107 FC?C 05
108 ISG 30
109 GTO 12
110♦LBL 14
111 FIX 2
112 999.99
113 X<>Y
114 RND
115 X>0?
116 X>Y?
117 XEQ 17
118 RTN
119♦LBL C
120 SF 03
121♦LBL D
122 FIX IND 38
123 SF 12
124 STO 28
125 3
126 FC? 03
127 4
128 STO 06
129 RCL 28
130 XROM "CV"
131 "IF X="
132 FC? 03
133 "IF Y="
134 ARCL 28
135 AVIEW
136 PSE
137 "Y="
138 FC?C 03
139 "X="
140 ARCL X
141 AVIEW
142 PSE
143 ADV
144 GTO 07
145♦LBL B
146 ENTER↑
147 2
148 GTO 11
149♦LBL E
150 5
151♦LBL 11
152 FIX IND 38
153 SF 12
154 STO 06
155 RDN
156 XROM "CV"
157 "1: LIN"
158 ASTO 01
159 "2: EXP"
160 ASTO 02
161 "3: LOG"
162 ASTO 03
163 "4: PWR"
164 ASTO 04
165 CLA
166 ARCL IND 07
167 AVIEW
168 PSE
169 "a="
170 ARCL 09
171 AVIEW
172 PSE
173 "b="
174 ARCL 08
175 AVIEW
176 PSE
177 "r="
178 ARCL 10
179 AVIEW
180 PSE
181 "r↑2="
182 RCL 10
183 X↑2
184 ARCL X
185 AVIEW
186 ADV
187♦LBL 07
188 FC? 55
189 RTN
190 "TO PLOT: R/S"
191 CF 12
192 PROMPT
193 "♦♦♦"
194 ASTO 26
195 "♦α    "
196 ASTO 27
197 "×QABQ♦"
198 ASTO 28
199 "♦×♦"
200 ASTO 31
201 7.011
202 ENTER↑
203 33.037
204 XROM "BE"
205 RCL 30
206 INT
207 STO Y
208 1
209 -
210 1 E-3
211 *
212 +
213 STO 24
214 FRC
215 39
216 +
217 STO 25
218 XROM "BX"
219 .01
220 *
221 STO 00
222 X<>Y
223 .01
224 *
225 STO 01
226 -
227 ABS
228 .25
229 *
230 ST+ 01
231 ST- 00
232 RCL 00
233 X<0?
234 0
235 STO 00
236 STO 04
237 RCL 01
238 -
239 -62
240 /
241 STO 32
242 "CRV"
243 ASTO 11
244 XEQ 16
245 RCL 25
246 XROM "S2"
247 STO 30
248 RCL 24
249 1
250 -
251 RCL IND X
252 INT
253 RCL 39
254 INT
255 -
256 RCL 29
257 /
258 STO 10
259 RCL 39
260 XEQ 00
261 X<>Y
262 X≠Y?
263 X>Y?
264 -
265 ABS
266 STO 08
267 RCL 24
268 1
269 -
270 RCL IND X
271 XEQ 00
272 3
273 *
274 +
275 STO 09
276 XROM "PRPLOTP"
277♦LBL "CRV"
278 FC?C 00
279 GTO 11
280 7.011
281 ENTER↑
282 33.037
283 XROM "BE"
284 XEQ 16
285 RCL 24
286 INT
287 .999
288 +
289 STO 30
290 FIX IND 38
291 BEEP
292 STOP
293♦LBL 11
294 FS? 02
295 GTO 08
296 STO [
297♦LBL 06
298 RCL IND 30
299 XEQ 00
300 2
301 /
302 RCL [
303 +
304 X>Y?
305 GTO 09
306 FS? 03
307 GTO 10
308 GTO 08
309♦LBL 00
310 INT
311 1 E2
312 /
313 RCL 10
314 RTN
315♦LBL 09
316 X<>Y
317 RCL [
318 RCL 10
319 2
320 /
321 -
322 X<=Y?
323 GTO 11
324 FS? 03
325 GTO 10
326♦LBL 08
327 RCL 31
328 STO 03
329 FS?C 02
330 SF 00
331 RCL [
332 GTO IND 33
333♦LBL 11
334 FS? 03
335 GTO 08
336 SF 03
337 ISG 30
338 GTO 06
339 SF 02
340♦LBL 10
341 1
342 ST- 30
343 CF 03
344 RCL 26
345 FS?C 01
346 GTO 15
347 RCL 27
348 FS?C 04
349 GTO 15
350 RCL 28
351♦LBL 15
352 CF 04
353 STO 03
354 RCL IND 30
355 FRC
356 1 E3
357 *
358 ISG 30
359 RTN
360 RTN
361♦LBL 08
362 1
363 ST- 30
364 RCL IND 30
365 FRC
366 ISG 30
367 RCL IND 30
368 FRC
369 -
370 1 E3
371 *
372 ABS
373 RCL 32
374 ISG 30
375 SF 04
376 X<Y?
377 SF 01
378 GTO 06
379♦LBL 02
380 RCL 34
381 *
382 E↑X
383 RCL 35
384 *
385 RTN
386♦LBL 03
387 LN
388♦LBL 01
389 RCL 34
390 *
391 RCL 35
392 +
393 RTN
394♦LBL 04
395 RCL 34
396 Y↑X
397 RCL 35
398 *
399 RTN
400♦LBL 16
401 RCL 25
402 STO 30
403♦LBL 05
404 RCL IND 30
405 STO Z
406 FRC
407 1 E5
408 *
409 STO Y
410 RCL Z
411 INT
412 1 E5
413 /
414 ST+ Y
415 RDN
416 STO IND 30
417 ISG 30
418 GTO 05
419 RTN
420♦LBL 17
421 FC? 24
422 FC? 55
423 RTN
424 0
425 /
426♦LBL b
427 1
428 -
429 100
430 *
431 STO 29
432 GTO 07
433♦LBL c
434 STO 38
435 GTO 12
436♦LBL d
437 SF 24
438 .END.
```

monthly basis and compute the equivalent monthly PMT.

Do:	See:	Result:
e	"DE"	Clear, Discrete/End status (PF=1 after clearing)
12 H	12.00	CF=12
10 A	10.00	n=10
10.5 B	10.50	NAR=10.5%=%I
5029.71 CHS D	-5,029.71	PMT=$5,029.71
C	29,595.88	PV=$29,595.88
12 I	12.00	PF=12, set monthly basis
10 a	120.00	n=120 (monthly)
D	-399.35	PMT=$399.35 (monthly)

Example 15: Perpetuity - Continuous Compounding
If you can purchase a single payment annuity with an initial investment of $60,000 that will be invested at 15% NAR compounded continuously, what is the maximum monthly return you can receive without reducing the $60,000 principal? If the interest rate is constant and the principal is not disturbed the payments can go on indefinitely (a perpetuity). Note that the term "n" of a perpetuity is immaterial. It can be any non-zero value. Set status to "CE".

Do:	See:	Result:
e	"DE"	Clear, Discrete/End status (CF=1 after clearing)
c	"CE"	Continuous/End status
12 A	12.00	n=12
I	12.00	PF=12
15 B	15.00	NAR=15%=%I
60000 E	60,000.00	FV=$60,000.00
CHS 1 X C	-60,000.00	Data entry flag is set so PV is stored as $60,000.00
D	754.71	PMT=$754.71

SUPPORTIVE PROGRAMS FOR FI

There are two optional routines provided below to extend the capability of the ROM routine FI . These routines are not located in the ROM, and must be loaded into RAM memory for their execution. They are named LPAS and FAST.

1. LBL LPAS

LBL LPAS "Loan Payments and Amortization Schedule" is really a full program in its own right, although it does use ROM routines FI , CJ , and CP . LPAS extends the capabilities of FI to accommodate "shifted" payment situations, when the first periodic payment does not fall at the beginning (BEGIN) or the end (END) of the first period, but at any date after the effective date. LPAS also provides an amortization schedule as an option.

2. LBL FAST - Reducing Interest Solution Time

LBL FAST is an optional routine used when solving for interest. Its purpose is to provide an initial starting guess for the interest-solving loop which is closer to the exact solution than that provided by LBL FI initial guess. The result is that interest solving execution time is usually shorter.

Don Dewey (5148) produced both supporting programs.

APPLICATION PROGRAM 1 FOR FI

LPAS - Loan Payments and Amortization Schedule

The FI program, like most financial programs and calculators, assumes that the first periodic payment occurs on either the first or last day of the payment period as specified by the beginning of period/end of period switch or toggle. Many financial agreements do not follow this convention. An agreement may call for the regular periodic payments to start earlier or later in order to provide a better match to other cash flow considerations of the borrower or lender. These agreements with "shifted" initial payment dates can be handled by conventional financial programs by computing an effective present value (PV) that compensates for the difference in interest accrued during the irregular first payment period. This computation becomes more complex when the compounding and payment frequencies (CF and PF) are unequal.

Shifting the initial payment date forces a change in the number or amount of the periodic payments or in the amount of the final or balloon payment. However, the participants to an agreement may want to specify the number and/or amount of the regular payments, and adjust the final payment to complete the amortization. Even without a shifted initial payment date or other restrictions the regular periodic payments seldom precisely complete the amortization and the final payment must be adjusted to accomplish this.

For the uninitiated or infrequent user of financial programs, the accomodation of a shifted first payment date and/or the computation of the correct final payment amount can cause problems. The following program easily handles these cases and also takes the drudgery out of computing an amortization schedule.

The LPAS program uses the FI program and the CJ and CP routines in the PPC ROM to expand the capabilities of the FI program to accomodate "shifted" initial payment dates and to compute the number and amount of periodic payments, and the final payment required to amortize a loan or to accumulate a specific future value. The information needed to prepare a loan amortization schedule may also be computed on an optional basis. The extensive capabilities of the FI program are used in their normal manner to define the parameters of a specific problem and to develop the initial solution. Two additional input parameters are provided; the effective date (ED) and the initial payment date (IP). These two dates define the length of the first payment period which need not be equal to the normal payment period implied by the payment frequency value (PF). The initial payment date (IP) also establishes the number of payments that will occur in the first year. The program computes the regular periodic payment and the final payment required to amortize a loan or to accumulate a specified future value over a specified term (n), or the number of payments and the final payment necessary to amortize a loan or to accumulate a specified future value with a specified periodic payment amount.

Conventional loans, mortgages with or without balloon payments, and Canadian or European mortgages are all acceptable to the LPAS program. Cases with payment frequencies of semi-monthly (PF=24) or less, use a 30 day month convention for determining the number of days of shift in the first payment date and the number of payments occurring in the first year. For payment frequencies greater than semi- monthly (i.e., daily, weekly, or bi-weekly) the acutal number of calendar days is used.

LPAS Program - Operation

The LPAS program computes the regular periodic payment and the final payment for both present value (PV*) and future value (FV*) cases. PV* cases involve periodic payments that reduce or amortize a present value. FV* cases involve appreciation or accumulation to a future value. The amortization schedule portion of the program supports PV* cases only. The LPAS program can be used with or without a printer (CF21).

The FI program is accessed and used to set the status (CB, CE, DB, DE), the compounding frequency (CF), the payment frequency (PF), the standard financial values (n, %i, PV, PMT, FV) and to solve for any missing financial value. Note: The FI program can be accessed by pressing "J" when the LPAS program has control. After entering the normal financial program data, the effective date of the financial agreement (ED) and the date of the initial payment (IP) are entered into the X and Y registers in the form MM.DDYYYY (Y=ED, X=IP). The IP date must not be earlier than the ED date.

The LPAS program is then executed. For easy access the LPAS program should be assigned to a key. The LPAS program was assigned to the X<>Y (F) key in the keystroke solutions in the example programs below. The program computes the regular periodic payment required to maintain the specified interest rate. The computation compensates for any fractional portion of the term and for any deviation from the normal initial payment date. When the program first stops, the computed payment (rounded to two decimal places) is in the PMT register (R04) and is displayed in the X register.

First Stop - The computed payment may be accepted, or a modified payment may be entered and substituted by pressing key "D". To continue the computation, select one of the following two options:

1. By pressing "H" the amortization period is limited to the integer portion of the term (n) and the final or balloon payment is adjusted to complete the amortization.

2. By pressing "J" the term (n) is recomputed to accomplish the amortization with the specified periodic payment with a minimum adjustment to the final or balloon payment.
The amortization choice restarts the program and the number of periodic payments and the amount of the final payment are computed. At the second stop the stack contains:

T = number of payments occuring in first year
Z = number of regular periodic payments
Y = amount of the regular periodic payment
X = amount of the combined final and balloon payments

Second Stop - An amortization schedule may be computed by pressing "E" (for PV* cases only) or control may be returned to the FI program by pressing "J".

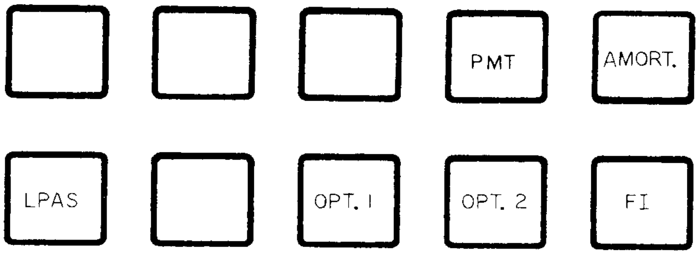
If an amortization schedule is computed and a printer is not available (FC?21) the program will stop after computing the values for each year. At each stop the stack will contain:

T = cumulative interest paid
Z = balance outstanding after last PMT for year
Y = interest paid during the year
X = year (YY)

To compute the amortization data for each succeeding year press R/S. Completion of the amortization is indicated by ** in the display. The total interest paid is in the Y register at this final stop.

After completion of the amortization, control may be returned to the FI program by pressing "J".

Keyboard Functions: (LPAS Program)



Key	Function	(Flag/Reg)
D	Enter revised periodic payment	"PMT" (R04)
E	Compute amortization data (PV* case only)	(R00-R13)
F	Enter ED and IP dates and compute periodic PMT	
H	Select Option 1 and compute final PMT	(F07)
I	Select Option 2 and compute term n and final PMT	
J	Transfer to FI program and display status	
R/S	Compute amortization data for next year	

Program requirements and limitations. CJ, CP, FI are the PPC ROM required routines. LPAS is 655 bytes SIZE=014 Flags 06-10,21,28,29

Acceptable Payment Frequencies (PF) are:

1 = annual	12 = monthly
2 = semi-annual	24 = semi-monthly
3 = tri-annual	26 = bi-weekly
4 = quarterly	52 = weekly
6 = bi-monthly	365 = daily

WARNING: Solutions using or resulting in a zero rate of interest (%i) will cause a "DATA ERROR".

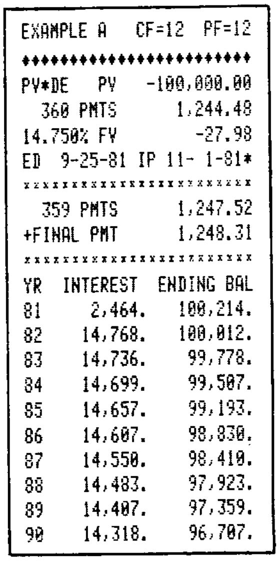
The output of LPAS is printed in three sections separated by horizontal lines. The first section records the original parameters of the case. The second section records the amount and number of regular payments and the final payment necessary to satisfy the options selected. The third section displays the optional amortization schedule.

Examples:

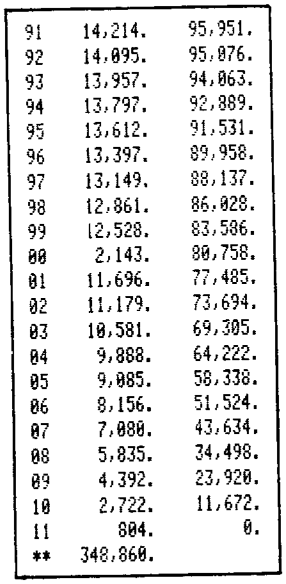
In the keystroke solution for each example, the lower case letters a through e represent shifted functions of keys A through E. Key in the indicated quantities and press the user defined keys as indicated in the "Do" column. Contents of the display or the printed output at significant points in the solution are shown in the "See" column and are followed by identification in the "Result" column. Use FIX 2 display mode, and assign LPAS to key F (X<>Y).

Example A: Conventional Mortgage. Develop the data for an amortization schedule for a fully amortized 30-year, $100,000 mortgage at 14.75% NAR compounded monthly with end of period payments of $1,244.48 with the first payment due on November 1, 1981. Effective date of the loan is September 25, 1981. Use option 1 (H) to limit the amortization period to 360 payments.

Do:	See:	Result:
CLX STO 06	0.00	Store FI function call
XEQ "FI"	"DE"	Discrete/End status
12 H I	12.00	CF=PF=12
30 a	360.00	n=360
14.75 B	14.75	NAR=14.75%=%I
100000 CHS C	-100,000.00	PV=$100,000.00
1244.48 D	1,244.48	PMT=$1,244.48
E	-27.98	FV=$27.98
9.251981		
ENTER↑		
11.011981 F	1,247.52	PMT, shifted IP $1,247.52
H	1,248.31	Final PMT=$1.248.31
E	**	Compute amortization data, see print out below.



```
EXAMPLE A   CF=12  PF=12
++++++++++++++++++++++++
PV*DE  PV   -100,000.00
  360 PMTS     1,244.48
14.750% FV       -27.98
ED  9-25-81 IP 11- 1-81*
xxxxxxxxxxxxxxxxxxxxxxxx
  359 PMTS     1,247.52
+FINAL PMT     1,248.31
xxxxxxxxxxxxxxxxxxxxxxxx
YR  INTEREST  ENDING BAL
81     2,464.   100,214.
82    14,768.   100,012.
83    14,736.    99,778.
84    14,699.    99,507.
85    14,657.    99,193.
86    14,607.    98,830.
87    14,550.    98,410.
88    14,483.    97,923.
89    14,407.    97,359.
90    14,318.    96,707.
```



```
91    14,214.    95,951.
92    14,095.    95,076.
93    13,957.    94,063.
94    13,797.    92,889.
95    13,612.    91,531.
96    13,397.    89,958.
97    13,149.    88,137.
98    12,861.    86,028.
99    12,528.    83,586.
00     2,143.    80,758.
01    11,696.    77,485.
02    11,179.    73,694.
03    10,581.    69,305.
04     9,888.    64,222.
05     9,085.    58,338.
06     8,156.    51,524.
07     7,080.    43,634.
08     5,835.    34,498.
09     4,392.    23,920.
10     2,722.    11,672.
11       804.        0.
**   348,860.
```

Note: 2 payments in 1981. The negative amortization during the first two years is due to the delayed first payment date. The asterisk following the IP date indicates a shifted initial payment date.

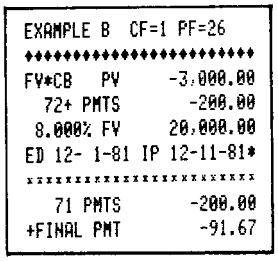
If a printer is not used when working Example A, after execution the stack will contain the following:

	after F		after H		after E*	
T=	-	T=	2.00	T =	2,464.	Σ Int.
Z=	-	Z=	359.00	Z =	100,214.	E.Bal.
Y=	-	Y=	1,247.52	Y=	2,464.	Yr.Int.
X=	1,247.52	X=	1,248.31	X=	81.	Year

*Press R/S to advance amortization to next year. the end of the amortization is indicated by ** in display.

Example B: Sinking Fund / Savings Plan Starting with an initial deposit of $3,000 compute the number of bi-weekly deposits of $200 and the amount of the final deposit needed to accumulate a balance of $20,000 in an account paying 8% compounded continuously, if the intial deposit (PV) is made on December 1, 1981 and the first bi-weekly deposit (PMT) is made on December 11, 1981. Set the status to CB.

Do:	See:	Result:
J	"DE"	Return to FI
c	"CE"	Set Continuous compounding
d	"CB"	Set Beginning of period payments
e	"CB"	Clear Financial Status=Continuous/Beginning CF=1 after clearing
26 I	26.00	PF=26
8 B	8.00	NAR=8%=%I
3000 CHS C	-3,000.00	PV=$3,000.00
200 CHS D	-200.00	PMT=$200.00
20000 E	20,000.00	FV=$20,000.00
A	72.43	n=72.43
12.011981		
ENTER↑		
12.111981 F	-197.29	PMT, shifted IP $197.29
200 CHS D	-200.00	Enter revised PMT $200.00
I	-91.67	Final PMT=$91.67
E	"FV* ?"	indicates attempted amortization of FV* case



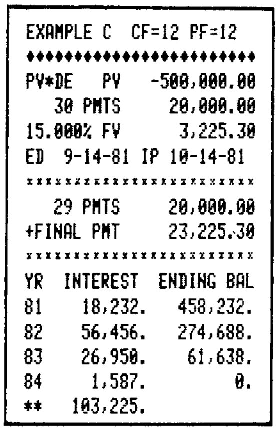
```
EXAMPLE B  CF=1 PF=26
++++++++++++++++++++++++
FV*CB   PV    -3,000.00
  72+ PMTS      -200.00
 8.000% FV    20,000.00
ED 12- 1-81 IP 12-11-81*
xxxxxxxxxxxxxxxxxxxxxxxx
   71 PMTS      -200.00
+FINAL PMT       -91.67
```

FV*CB = Future Value case with Continuous compounding and Beginning of period payments/deposits. The plus (+) sign following the number of payments indicates that the term includes a fractional payment period as developed from the original specifications.

Example C: Loan with Balloon Payment. Develop the amortization data for a $500,000 loan at 15% NAR with monthly compounding, to be repaid with 30 monthly end of period payments of $20,000 and a balloon payment of $3,225.30 coincident with the final payment. The loan effective date is September 14, 1981 and the first payment is scheduled for October 14, 1981.

Do:	See:	Result:
J	"CB"	Return to FI Status from previous example
c	"DB"	Set Discrete compounding
d	"DE"	Set End of period payments
e	"DE"	Clear Financial, final status= Discrete/End
12 H I	12.00	CF=PF=12
30 A	30.00	n=30
15 B	15.00	NAR=15%=%I
500000 CHS C		
	-500,000.00	PV=$500,000.00
20000 D	20,000.00	PMT=$20,000.00
E	3,225.30	Balloon=$3,225.30
9.141981		
ENTER↑		

10.141981 F	20,000.00	PMT=$20,000.00
H	23,225.30	Final + Balloon = $23,225.30
E	**	Compute amortization data, see print out below.

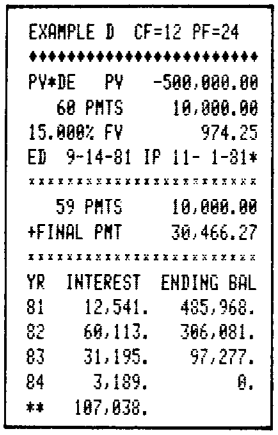


```
EXAMPLE C  CF=12 PF=12
♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦
PV*DE   PV   -500,000.00
   30 PMTS     20,000.00
15.000% FV      3,225.30
ED  9-14-81 IP 10-14-81
xxxxxxxxxxxxxxxxxxxxxxxx
   29 PMTS     20,000.00
+FINAL PMT     23,225.30
xxxxxxxxxxxxxxxxxxxxxxxx
YR  INTEREST  ENDING BAL
81    18,232.   458,232.
82    56,456.   274,688.
83    26,950.    61,638.
84     1,587.         0.
**   103,225.
```

Because the initial payment occurs exactly one month after the loan effective date there is no change in the re-computed PMT.

Example D: Delayed First Payment This example will illustrate the effect of a different repayment plan for the loan defined in Example C. Develop the data for amortizing a $500,000 loan at 15% NAR with monthly compounding, to be repaid with 60 semi-monthly end of period payments of $10,000 and a balloon payment coincident with the final payment. The loan effective date is September 14, 1981 and the first payment is scheduled for November 1, 1981.

Do:	See:	Result:
J	"DE"	Return to FI Status left from Example C
12 H	12.00	CF=12
24 I	24.00	PF=24
60 A	60.00	n=60
15 B	15.00	NAR=15%=%I
500000 CHS C	-500,000.00	PV=$500,000.00
10000 D	10,000.00	PMT=$10,000
E	974.25	FV=$974.25
9.141981		
ENTER↑		
11.011981 F	10,268.92	PMT=$10,268.92
10000 D	10,000.00	Set PMT=$10,000.00 exactly
H	30,466.27	Final+Balloon=$30,466.27
E	**	Compute amortization data See print out below



```
EXAMPLE D  CF=12 PF=24
♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦
PV*DE   PV   -500,000.00
   60 PMTS     10,000.00
15.000% FV        974.25
ED  9-14-81 IP 11- 1-81*
xxxxxxxxxxxxxxxxxxxxxxxx
   59 PMTS     10,000.00
+FINAL PMT     30,466.27
xxxxxxxxxxxxxxxxxxxxxxxx
YR  INTEREST  ENDING BAL
81    12,541.   485,968.
82    60,113.   306,081.
83    31,195.    97,277.
84     3,189.         0.
**   107,038.
```

The total interest on this repayment plan is $3,813 more than in Example C due to the delayed first payment date and the smaller payments. The borrower has the use of more money for a longer time.

LPAS Program - Equations

All equations assume the use of standard financial transaction sign conventions of money received as positive (+) and money paid out as negative (-).

Notation used:

d	= number of days in payment period
I_e	= effective interest rate per payment period
m	= integer portion of term n
n	= number of payment periods in term
s	= number of days first payment is shifted
CF	= compounding frequency per year
ED#	= effective date - day number
FV	= future value after n periods
FV_m	= future value after m periods
FV_{m-1}	= future value after m-1 periods
FV*	= future value case
INT	= interest for the year
IP#	= initial payment date - day number
NP	= number of payments in the year
PF	= payment frequency per year
PMT	= periodic payment
PMT_f	= final payment
PV	= present value
PV_e	= effective present value
PV*	= present value case

If $|FV| <= |PV|$, then PV* case
If $|FV| > |PV|$, then FV* case

The initial payment date is "shifted" when: $s \neq 0$

where: $s = IP\# - ED\#$ for beginning of period payments

$s = IP\# - ED\# + d$ for end of period payments

For financial calculations involving a "shifted" first payment date, the present value (PV) must be converted to an effective present value (PV_e) that is adjusted to compensate for the difference in interest accrued during the irregular first payment period.

$$PV_e = PV(1+I_e)^{(sPF/dCF)}$$

To precisely complete the amortization of a present value or the accrual of a future value, the final payment must be calculated separately from the regular periodic payment. The LPAS program incorporates eight variations of final payment calculations.

$PMT_f = FV_{m-1}$	PV* case, annuity due, Option 1
$= FV_m$	PV* case, annuity due, Option 2
$= FV_m + PMT$	PV* case, ordinary annuity, Option 1

$PMT_f = FV_m(1+I_e)$	PV* case, ordinary annuity Option 2
$= FV_{m-1} - FV/(1+I_e)$	FV* case, annuity due, Option 1
$= FV_m - FV/(1+I_e)$	FV* case, annuity due Option 2
$= FV_m + PMT - FV$	FV* case, ordinary annuity Option 1
$= FV_m(1+I_e) - FV$	FV* case, ordinary annuity Option 2

N.B. Values m and n are different for Options 1 and 2

The interest paid during each year of amortization is determined by the difference between the ending and beginning balances plus the sum of the payments for the year.

INT = (NP*PMT) + PV + FV

LPAS Program - Line by Line Analysis

LBL LPAS - Mainline - First Section

```
001   store ED and IP dates.  Print separator line
007   set flag F06 (FV* case) if: |FV|>|PV|
014   calculate term (n).   Print line 1           (PV)
026   Format data.          Print line 2       (n)(PMT)
038   Format date.          Print line 3       ($I)(FV)
044   If PF>24 set Flag 07 (calendar year basis)
050   Calculate day number of effective date     (ED#)
053   Calculate day number of first PMT date     (IP#)
056   Develop number of days from ED thru IP date  (s)
058   Develop number of day from IP thru year end
069   Develop number of days in normal PMT period  (d)
079   Adjust s for end of period payments          (s)
081   Develop number of payments in first year
086   If PMT = 0, set s = 0
090   If s ≠ 0, append *.  Print line 4        (ED)(IP)
095   Develop PVe to adjust for shifted IP date  (PVe)
107   Save IP year (YY) and calculate payment    (PMT)
113   --FIRST STOP--
```

At this stop the calculated periodic payment may be accepted or the original or a modified payment can be entered and stored by pressing key D before selecting an amortization option (H or I).

```
      Subroutines
LBL 01 Reformat date for CJ and load print buffer
LBL 02 Calculate day number using 30/360 convention
LBL 03 Calculate day number using CJ (calendar
         basis)
LBL 04 Format month (MM) and day (DD) for printing
LBL 05 Display control -
    06                 - and column format subroutine
LBL 07 Execute specified FI routine
LBL 08 Fill buffer with specified character -
    09                   - and printer separator line
LBL D  Store PMT in R04
LBL J  Transfer control to FI and display status
```

```
 - Mainline - Second Section
LBL H  Option 1 - If PMT≠0, set flag F07 (set=opt. 1)
LBL I  Option 2
205    Calculate new (n). If n=0 use original n   (n)
213    Select (n): option 1=original  option 2=new
216    Calculate FV and modify to -
LBL 10       - develop final payment          (PMTf)
LBL 11 Store final PMTf   Print separator line
263    Format data.        Print line 5   (n-1)(PMT)
269    Format data.        Print line 6       (PMTf)
278    --SECOND STOP--
```

At this stop the amortization schedule calculation may be selected by pressing key E (for PV* cases only), or control may be returned to the FI program via key J.

```
       Subroutine
LBL 12 Format control subroutine

LBL E  - Mainline - Third Section - Amortization
284     If FV* case stop and display "FV* ?" (invalid)
288     Print separator line.  Print heading line
294     Reduce payment count by number 1st yr payments
LBL 13 Develop interest for year -
    14                    - and calculate ending balance
LBL 15 ΣINT and format data.  Print amortization line
346     Load stack for review and stop if FC?21
351     --AMORTIZATION YEAR STOP--
```

If flag F21 is cleared this stop will occur after the amortization calculations have been made for each year. Amortization data is available in the stack.

```
352     If not final year, update year & payment count
LBL 16 End routine         Print total       (**)( ΣINT)
384     END
```

Other LPAS program technical details:

Global Label: LPAS
Local Labels: D,E,H,I,J, and 01-16
Byte Count: 655 (requires one memory module)
Size Required: SIZE=014
ROM Routines called: CJ, CP, FI
Subroutine Levels: 3
Flags Used: LPAS - 06,07,21,28,29
FI - 08, 09, & 10
CJ - 10
CP - 29 & 40

Data Registers Used:

R00: multi use store	R07: I as decimal
R01: n term	R08: CF compounding freq.
R02: $I as percentage	R09: PF payment frequency
R03: PV present value	R10: multi use store
R04: PMT periodic pmt	R11: multi use store
R05: FV future value	R12: multi use store
R06: IND addr.	R13: multi use store

Status Registers: none used
Alpha Registers: all used
Σ REG: not used
Peripherals: printer recommended but not required
Stack Usage: I/O see program description
Execution Time: variable

BAR CODE ON PAGE 480

APPLICATION PROGRAM FOR: **FI**

```
01♦LBL "LPAS"
02 STO 10
03 X<>Y
04 STO 00
05 0
06 XEQ 08
07 CF 06
08 RCL 03
09 ABS
10 RCL 05
11 ABS
12 X>Y?
13 SF 06
14 1
15 XEQ 07
16 ASTO X
17 "P"
18 FS? 06
19 "F"
20 "⊦V*"
21 ARCL X
22 "⊦   PV"
23 RCL 03
24 XEQ 05
25 ADV
26 XEQ 12
27 RCL 01
28 ENTER↑
29 INT
30 ARCL X
31 -
32 X≠0?
33 "⊦+"
34 "⊦ PMTS"
35 RCL 04
36 XEQ 05
37 ADV
38 FIX 3
39 ARCL 02
40 "⊦% FV"
41 RCL 05
42 XEQ 05
43 ADV
44 XEQ 12
45 CF 07
46 24
47 RCL 09
48 X>Y?
49 SF 07
50 "ED "
51 RCL 00
52 XEQ 01
53 X<> 10
54 "⊦ IP "
55 XEQ 01
56 ST- 10
57 STO 00
58 FIX 2
59 SF 28
60 SF 29
61 1
62 ST+ 11
63 STO 12
64 CLX
65 STO 13
66 XEQ 02
67 RCL 00
68 -
69 360
70 ENTER↑
71 6
72 FC?C 07
73 CLX
```

```
74 +
75 RCL 09
76 /
77 INT
78 STO 13
79 FC? 09
80 ST+ 10
81 +
82 LASTX
83 /
84 INT
85 STO 12
86 RCL 10
87 RCL 04
88 X≠0?
89 X<>Y
90 CHS
91 X≠0?
92 "⊦*"
93 FS? 21
94 PRA
95 CLA
96 RCL 08
97 RCL 13
98 *
99 /
100 RCL 09
101 *
102 RCL 07
103 LN1+X
104 *
105 E↑X
106 ST* 03
107 RCL 06
108 STO 11
109 4
110 XEQ 07
111 RND
112 STO 04
113 RTN
114♦LBL 01
115 INT
116 -100
117 STO 11
118 STO Z
119 X<>Y
120 STO 12
121 XEQ 04
122 INT
123 STO 13
124 XEQ 04
125 CHS
126 ST* 11
127 FRC
128 *
129 STO 06
130 10
131 X>Y?
132 "⊦0"
133 ARCL Y
134♦LBL 02
135 FS? 07
136 GTO 03
137 RCL 11
138 360
139 *
140 RCL 12
141 30
142 *
143 +
144 RCL 13
145 +
146 RTN
```

APPLICATION PROGRAM FOR: **FI**

```
147♦LBL 03
148 RCL 11
149 RCL 12
150 RCL 13
151 XROM "CJ"
152 RTN
153♦LBL 04
154 10
155 X>Y?
156 "⊦ "
157 ARCL Y
158 RDN
159 LASTX
160 -
161 *
162 "⊦-"
163 RTN
164♦LBL 05
165 FIX 2
166 9
167♦LBL 06
168 STO 06
169 X<>Y
170 SF 28
171 SF 29
172 FC? 21
173 RTN
174 ACA
175 XROM "CP"
176 CLA
177 RTN
178♦LBL 07
179 STO 06
180 XROM "FI"
181 RTN
182♦LBL 08
183 FC? 21
184 RTN
185 24
186 X<>Y
187♦LBL 09
188 ACCHR
189 DSE Y
190 GTO 09
191 PRBUF
192 RTN
193♦LBL D
194 STO 04
195 RTN
196♦LBL J
197 10
198 STO 06
199 GTO "FI"
200♦LBL H
201 RCL 04
202 X≠0?
203 SF 07
204♦LBL I
205 RCL 01
206 INT
207 STO 10
208 1
209 XEQ 07
210 INT
211 X=0?
212 RCL 10
213 FC? 07
214 STO 10
215 RCL 10
216 1
217 FS? 07
218 ST- 10
219 FS? 07
```

```
220 FC? 09
221 CLX
222 -
223 STO 01
224 RCL 07
225 1
226 +
227 STO 13
228 RCL 05
229 STO 00
230 5
231 XEQ 07
232 RCL 00
233 STO 05
234 FC? 06
235 CLX
236 STO 00
237 X<>Y
238 RCL 13
239 FS? 09
240 ST/ 00
241 X<>Y
242 FC? 09
243 GTO 10
244 RCL 00
245 -
246 GTO 11
247♦LBL 10
248 FC? 07
249 *
250 RCL 00
251 FC? 06
252 CLX
253 -
254 RCL 04
255 FC? 07
256 CLX
257 +
258♦LBL 11
259 RND
260 STO 13
261 1
262 XEQ 08
263 XEQ 12
264 ARCL 10
265 "⊦ PMTS"
266 RCL 04
267 XEQ 05
268 ADV
269 "+FINAL PMT"
270 RCL 13
271 XEQ 05
272 ADV
273 RCL 12
274 RCL 10
275 RCL 04
276 RCL 13
277 FIX 2
278 RTN
279♦LBL 12
280 FIX 0
281 CF 28
282 CF 29
283 RTN
284♦LBL E
285 "FV* ?"
286 FS? 06
287 PROMPT
288 1
289 XEQ 08
290 "YR  INTEREST "
291 "⊦ENDING BAL"
292 FS? 21
```

Listing continued on page 158.

Listing continued from 157.

BAR CODE ON PAGE 480

APPLICATION PROGRAM FOR: FI

```
293 PRA
294 CF 07
295 CLA
296 CLX
297 X<> 12
298 RCL 10
299 X<=Y?
300 SF 07
301 X<>Y
302 STO 01
303 -
304 STO 10
305♦LBL 13
306 XEQ 12
307 RCL 11
308 10
309 X>Y?
310 "⊢0"
311 ARCL Y
312 RCL 03
313 RCL 04
314 RCL 01
315 *
316 +
317 STO 00
318 CLX
319 STO 05
320 FC? 07
321 GTO 14
322 RCL 13
323 ST+ 00
324 GTO 15
325♦LBL 14
326 5
327 XEQ 07
328 FIX 2
329 RND
330 ST+ 00
331 STO 05
332 CHS
333 STO 03
334♦LBL 15
335 FIX 0
336 RCL 00
337 RND
338 ST+ 12
339 8
340 XEQ 06
341 RCL 05
342 RND
343 8
344 XEQ 06
345 ADV
346 RCL 12
347 RCL 05
348 RCL 00
349 RCL 11
350 FC? 21
351 STOP
352 FS?C 07
353 GTO 16
354 1 E2
355 RCL 11
356 1
357 +
358 X=Y?
359 -
360 STO 11
361 RCL 10
362 STO 01
363 RCL 09
364 ST- 10
365 X<=Y?
366 STO 01
367 -
368 X<=0?
369 SF 07
370 GTO 13
371♦LBL 16
372 "**"
373 RCL 12
374 8
375 XEQ 06
376 11
377 FS? 21
378 SKPCHR
379 ADV
380 "**"
381 ASTO X
382 FIX 2
383 .END.
```

APPLICATION PROGRAM 2 FOR FI

FAST - Reducing Interest Solution Time

When the solution for interest is required for PMT≠0, LBL 02 of FI produces an initial guess for the interest which is supplied to the iterative loop starting at LBL 06. In most cases the LBL 02 guess is usually "close" (in the mathematical sense) to the actual solution insuring that the interest solution is found in a reasonably short time.

Unfortunately, there will always exist a problem which will cause the LBL 02 guess to be far enough away from the actual solution to cause the execution time to be long. The optional routine presented below will provide an initial guess which tends to be "closer" to the actual solution than that provided by LBL 02, allowing a shorter execution time for most problems.

In use, the optional routine is executed in RAM memory and produces an initial guess for the interest. The guess is stored in register R07, and control of the calculator is transferred from the FAST routine to LBL 06 of the ROM program FI.

For the condition when PMT=0, the routine transfers to LBL 09 of the ROM program for an explicit solution. When solving for n, PV, PMT, or FV, the ROM is used in the usual manner. Don Dewey (5148) produced the mathematical expressions and wrote the program.

LBL FAST INSTRUCTIONS

1. Load the routine below into the calculator memory.

2. Go to LBL FI in the ROM.

3. Select desired status and enter known variables in the usual manner.

4. Either a) or b):

   a) solve for n, i, PV, PMT, or FV in the usual manner.

   b) Execute FAST to solve for interest using the optional routine. Do not use LBL B. The interest value is returned in the usual manner.

5. Repeat as needed from step 2.

APPLICATION PROGRAM FOR: FI

```
01♦LBL "FAST"
02 9
03 STO 06
04 RCL 04
05 X=0?
06 GTO "FI"
07 6
08 STO 06
09 RCL 05
10 RCL 04
11 RCL 01
12 *
13 -
14 LASTX
15 RCL 05
16 +
17 RCL 03
18 +
19 RCL 01
20 RCL 03
21 *
22 X≠0?
23 /
24 ABS
25 STO 07
26 X<>Y
27 RCL 01
28 1
29 -
30 X↑2
31 RCL 04
32 *
33 RCL 05
34 -
35 RCL 03
36 +
37 3
38 *
39 /
40 ABS
41 RCL 05
42 X=0?
43 GTO "FI"
44 RCL 04
45 *
46 X>0?
47 GTO "FI"
48 RDN
49 STO 07
50 GTO "FI"
51 .END.
```

EQUATIONS USED IN FAST ROUTINE

If PMT*FV < 0 then FV case.
If PMT*FV >= 0 then PV case.

1. PV CASE:

$$i_0 = \left| \frac{n*PMT + PV + FV}{n*PV} \right|$$

Problem valid only if PV*PMT < 0.

2. FV CASE:

a) For PV≠0:

$$i_0 = \left| \frac{FV - n*PMT}{3*[(n-1)^2*PMT + PV - FV]} \right|$$

b) For PV=0:

$$i_0 = \left| \frac{FV + n*PMT}{3*[(n-1)^2*PMT + PV - FV]} \right|$$

FORMULAS USED IN FI

The basic financial equation used in this program was first reported in the Hewlett-Packard Journal of October 1977 (Ref. 3) where the description of its implementation in the HP-92 Financial Calculator was given. In this unique equation, all five financial variables (n, i, PV, PMT, FV) are accounted for, using the simple rule that money paid out is considered negative in sign, while money received is considered positive in sign.

The equation from page 23 of Ref. 3, is:

(1) $PV*(1+i)^n + PMT*[(1+i)^n - 1]/i + FV = 0$

Ordinary Annuity and Annuity Due Selection

In its present form, equation (1) is suitable for the ordinary annuity condition, when payments are made at the end of each period. To enable (1) to solve the annuity due condition when payments are made at the beginning of each period, a small modification is required. When this modification is added, equation (1) becomes:

(2) $PV*(1+i)^n + PMT*(1+iX)*[(1+i)^n - 1]/i + FV = 0$

where X=0 for ordinary annuity condition
X=1 for annuity due condition

When flag F09 is cleared, the ordinary annuity condition is selected. When flag F09 is set, the annuity due condition is selected. Flag F09 is toggled by LBL d.

With a simple algebraic rearrangement, (2) becomes:

(3) $[PV+PMT(1+iX)/i][(1+i)^n-1] + PV + FV = 0$

or

(4) $(PV + C)A + PV + FV = 0$

where

(5) $A = (1+i)^n - 1$

(6) $B = (1+iX)/i$

(7) $C = PMT*B$

The form of equation (4) simplifies the calculation procedure for all five variables, which are readily solved as follows:

(8) $n = LN[(C-FV)/(C+PV)]/LN(1+i)$

n is solved using LBL 01

(9) $i = [FV/PV]^{1/n} - 1$

For PMT=0, i is solved using LBL 09

For PMT≠0, i must be solved by iteration

(10) $PV = -[FV + (A*C)]/(A+1)$

PV is solved using LBL 03

(11) $PMT = -[FV + PV(A+1)]/(A*B)$

PMT is solved using LBL 04

(12) $FV = -[PV + A(PV + C)]$

FV is solved using LBL 05

Solution of Interest When PMT≠0

To solve for interest i when PMT≠0, an iterative technique must be employed, as equation (1) cannot be explicitly solved for i. This program uses Newton's Method, using exact expressions for the function of i and its derivative. The expressions are:

(13) $i_{k+1} = i_k - f(i_k)/f'(i_k)$

where

(14) $f(i) = A(PV+C) + PV + FV$

(15) $f'(i) = n*D*(PV+C) - (A*C)/i$

where

(16) $D = (1+i)^{n-1}$

(17) $= (A+1)/(1+i)$ as calculated by LBL 06

The iterative interest solving loop using equations (13), (14), and (15) starts at LBL 06.

Starting Guess For Interest

To solve for interest using Newton's Method, an initial starting guess must be provided. The program uses the following expression to provide the initial guess, i_0:

(18) $$i_0 = \left| \frac{PMT}{|PV| + |FV|} \right| + \left| \frac{|PV| + |FV|}{n^3*PMT} \right|$$

The closer the initial guess i_0 is to the actual solution i, the greater is the probability that the required solution will be obtained, and the shorter is the execution time.

Further Program Refinements

As well as being able to select either an ordinary annuity or annuity due situation, the program also enables solutions to be obtained when

This routine was run beginning with 1 in the Y, Z and T registers and with X clear. R/S was pressed, and then pressed again after 100 seconds to establish a speed count. Results ranged from the low 1600's to middle 1700's for various 41C's, so 1700 was established as a reference count. Execution times presented for each example have been normallized to the 1700 speed count. If you have sped up your HP41, you should expect significantly faster execution times than reported below.

The following relationship was obtained for the HP routine, using nonlinear regression analysis:

Execution
time, min = -0.8905 + 0.1952*L + 0.3615*L*F

where L = Number of printed lines in the plot, and
F = Number of functions plotted simultaneously

This relationship holds for a 1700-count HP41C. Program HPT has been provided for the estimation of run times for HP plots, due to the wide range of times possible. This program will calculate estimated run times normallized to any count in the 100-second speed count test above and then executes HP. If the speed count is not known for the particular 41C being used, then simply pressing R/S at the appropriate time will assume a reference count of 1700.

Enter parameters for HP into data registers, including the number of functions in X, and then:

KEYSTROKES	DISPLAY	RESULT
XEQ HPT	COUNT?	Prompts for count
Enter count, or just press R/S for 1700 count time		Prints "EST RUN TIME:" and time, then runs HP

The listing for program HPT:

BAR CODE ON PAGE 480

APPLICATION PROGRAM FOR: HP

```
01♦LBL "MPT"
02 SF 08
03 GTO 00
04♦LBL "HPT"
05 CF 08            Store # functions
06♦LBL 00           plotted in R03
07 STO 03
08 1700
09 "COUNT?"
10 PROMPT           Store 1700 or count
11 STO 04           in R04
12 RCL 09
13 RCL 08
14 -
15 RCL 10           Compute the number
16 /                of lines to be
17 1                plotted
18 +
19 FS? 08
20 GTO 01
21 11
22 /
23♦LBL 01           Calculate estimated
24 RCL X            run time for HP
25 RCL 03           or MP
26 *
27 FS? 08
28 .09566
29 FC? 08
30 .3615
31 *
32 X<>Y
33 FS? 08
34 -.02144
35 FC? 08
36 .1952
37 *
38 +
39 FS? 08
40 .02516
41 FC? 08
42 -.8905
43 +
44 1700
45 *
46 RCL 04
47 /
48 "EST RUN TIME:"
49 "⊦ "             Print run time
50 FIX 2
51 ARCL X
52 "⊦ MIN."
53 PRA
54 RCL 03
55 FS? 08
56 XROM "MP"        Call HP or MP
57 FC? 08
58 XROM "HP"
59 RTN
60 .END.
```

The listing for HPT appears in section A.3 of the MP routine writeup, since HPT is a subset of program MPT, which performs timing for the MP routine in a similar fashion. The barcode for HPT/MPT appears in Appendix N.

A.4. Changing Display Annunciators.

As part of the operation of HP, flag 55 (the printer existence flag) is synthetically cleared using the IF routine in order to trick the calculator into assuming that no printer is present. This speeds up non-printing operations some 20 percent, which is significant in a plot that may take several minutes to complete. During the execution of the IF routine, the display annunciators may change, such as 'RAD' coming on, or flag annunciators going on or off. This situation will remain until the HP routine stops. If the user halts execution prematurely, the annunciators will return to their original configuration. This will also reset flag 55, since the printer will now be detected to be present. Pressing R/S to restart will eventually cause HP to detect that F55 is set, and again call the IF routine to clear it, and annunciators will again change. No changes will have actually occurred to flags or to any modes.

B. Variable Plot Width.

The plot width in columns is stored by the user in register R02. This can vary from 1 to 168 columns. This feature will be used extensively in many of the examples below.

C. Skip Standard Header.

If flag 07 is clear, a standard set of initial header lines is printed before the plot. This consists of each function name and its corresponding function identifier, plus the limits in the Y and X directions along with the X increment value. Setting flag 07 causes HP to skip the header information entirely and just print the Y axis, whether it is the standard 12 dashes or a user-defined axis (to be described later). This allows another header to be substituted and printed immediately before HP is called, if the user desires.

Figure 16. Plot of the Y=sinX function of Example 14 using HP. Three axes have also been plotted by storing the constants 1, 0 and -1 into the function name registers R16 to R18. Execution time: 17 min 55 sec.

H. Prompting for User Inputs to HP.

Because of the large number of inputs to the HP routine, it may be inconvenient to remember where all the input information belongs. The following program provides some assistance by prompting the user for all the basic inputs to HP: function names, Ymin, Ymax, plot width, Xmin, Xmax, and X increment. It then calls the HP routine. Simply set all the flags to their correct status, set the other options appropriately and XEQ HPP. The listing is presented below:

BAR CODE ON PAGE 479

APPLICATION PROGRAM FOR: HP

```
01♦LBL "MPP"
02 SF 08
03 GTO 00
04♦LBL "HPP"
05 CF 08
06♦LBL 00
07 "NO. FCNS?"        Input # of functions
08 PROMPT
09 STO 04
10 1 E3
11 /
12 15.014
13 +
14 STO 03
15 FIX 0
16♦LBL 01
17 "NAME "            Input each name or
18 RCL 03             X axis value
19 14
20 -
21 ARCL X
22 "⊢?"
23 AON
24 PROMPT
25 FS? 48
26 ASTO IND 03
27 FC? 48
28 STO IND 03
29 ISG 03
30 GTO 01
31 "Y MIN?"           Input Ymin
32 PROMPT
33 STO 00
34 "Y MAX?"           Input Ymax
35 PROMPT
36 STO 01
37 "PLOT WIDTH?"      Input plot width
38 PROMPT
39 STO 02
40 "X MIN?"           Input Xmin
41 PROMPT
42 STO 08
43 "X MAX?"           Input Xmax
44 PROMPT
45 STO 09
46 "X INC?"           Input X increment
47 PROMPT
48 STO 10
49 RCL 04
50 FIX 4
51 FS? 08
52 XEQ "MP"           Calls MP or HP
53 FC? 08
54 XEQ "HPT"
55 RTN
56 .END.
```

This routine may also be used for passing input to MP by pressing XEQ MPP. In that case, MP would be executed as the final step. If estimated execution times are also desired, one could replace the lines XROM HP and XROM MP with XEQ HPT and XEQ MPT respectively. Then, after all prompting, the run time would be printed before the plot routine was executed.

The barcode for HPP/MPP appears in Appendix N.

I. Plots using Multiple Paper Widths - 'Superplotting'.

When higher plot resolution is desired in the Y direction (across the printer paper) than can be obtained with 168 columns, it is possible to plot graphs with HP which require multiple widths of printer paper. This has been referred to as 'superplotting'. The routine shown below takes care of the housekeeping involved in printing each section of the plot, re-initializes the inputs and increments the Y limits. The only difference between the inputs for this program and for HP is that Ymax is stored in R42 instead of R01, and a Y increment value (the desired width of each printed plot section) is stored in R43. After all the function names are stored, simply set the limits and XEQ SHP:

1. Place the function names (and axis values) in R15 and up
2. Set disappearing overflow mode (CF05, SF06) so functions jump from strip to strip
3. Store Xmin, Xmax and Xinc in R08, R09 and R10
4. Store plot width in R02
5. Store Ymin in R00, Ymax in R42 and Yinc in R43
6. Enter the number of functions to be plotted
7. XEQSHP, and the plot is printed, a strip at a time, moving from Ymin to Ymax, in steps equal to the Y increment stored in R43.

The SHP listing is as follows:

BAR CODE ON PAGE 479

APPLICATION PROGRAM FOR: HP

```
01♦LBL "SMP"          MP superplotting
02 STO 38             Save # fcns in R38
03 RCL 08
04 STO 37             Ymin in R37
05 RCL 00
06 RCL 36
07 +
08 STO 01             Ymin + Y increment
09♦LBL 00
10 RCL 38             Restore # fcns
11 XEQ "MP"           Call to MP
12 RCL 01
13 RCL 35
14 X<=Y?
15 RTN                If done, stop
16 RDN
17 STO 00
18 RCL 36             If not, increment
19 ST+ 01             Ymin, Ymax
20 RCL 37
21 STO 08
22 GTO 00
23♦LBL "SHP"          HP superplotting
24 STO 45             Save # fcns in R45
25 RCL 08
26 STO 44             X min in R44
27 RCL 00
28 RCL 43
29 +
30 STO 01             Ymin + Y increment
31♦LBL 01
32 RCL 45             Restore # fcns
33 XEQ "HP"           Call to HP
```

```
34 RCL 01
35 RCL 42
36 X<=Y?        If done, stop
37 RTN
38 RDN
39 STO 00
40 RCL 43
41 ST+ 01       If not, increment
42 RCL 44       Ymin, Ymax
43 STO 00
44 GTO 01
45 END
```

Note that the SHP program listing also includes SMP, which is the superplotting routine for MP. See the MP writeup elsewhere in this manual. The barcode for SHP/SMP appears in Appendix N.

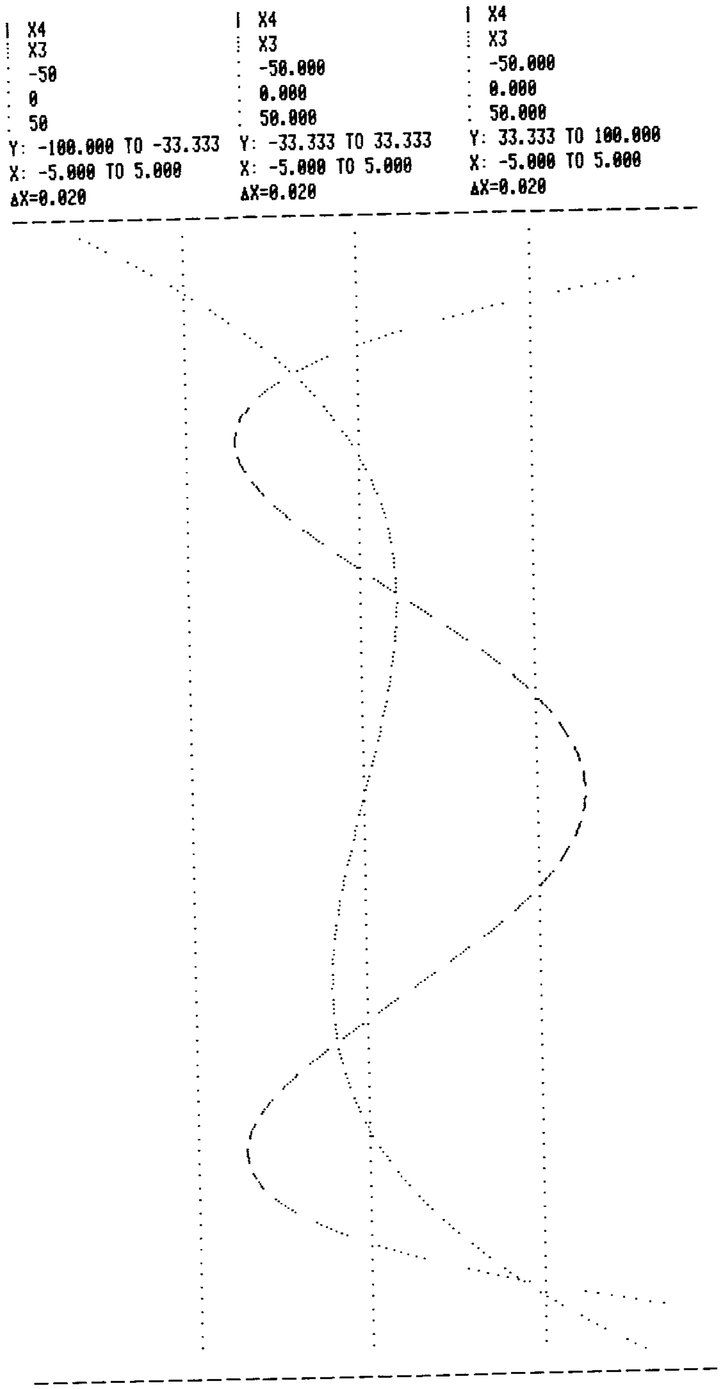
The first plot strip has Ymin = Ymin and Ymax = Ymin + Yinc. The next strip has Ymin = the previous Ymax and Ymax = (new Ymin) + Yinc. This process repeats until the current Ymax exceeds that which was stored in R42. If Yinc is not chosen properly, the last plot strip will exceed the designated upper limit in the Y direction, but the excess may be removed by the user with a scissors if so desired.

Example 15. Use HP superplotting to plot the following 2 functions: Y=X↑4 - 20*X↑2 + 64 and Y = X↑3 - 9X simultaneously. Use Y limits of -100 and +100 with a Y increment of 66.67 (3 strips wide). Let the X limits be -5 and +5 with an X increment of 0.02. Use function identifiers #1 and #2 for the 2 functions and also plot X axes at Y=-3, Y=0 and Y=+3 using identifier #3 for each.

APPLICATION PROGRAM FOR: **HP**

```
01♦LBL "X4"     Function #1
02 STO Y
03 4
04 Y↑X
05 X<>Y
06 X↑2
07 20
08 *
09 -
10 64
11 +
12 RTN
13♦LBL "X3"     Function #2
14 STO Y
15 3
16 Y↑X
17 X<>Y
18 9
19 *
20 -
21 RTN
22♦LBL "SP2"    Plot routine
23 XROM "RF"
24 SF 21        Clear F00 - F28,
25 55           SF21, SF55
26 XROM "IF"
27 SF 06        Set disappearing mode
28 "X4"
29 ASTO 15
30 "X3"
31 ASTO 16      Store fcn names
32 -50
33 STO 17
34 0
35 STO 18
36 50
37 STO 19
38 .12444
39 STO 12       Store symbol map
```

```
40 SF 04
41 -5
42 STO 08       Xmin
43 CHS
44 STO 09       Xmax
45 .02
46 STO 10       X increment
47 168
48 STO 02       Plot width
49 -100
50 STO 00       Ymin
51 CHS
52 STO 42       Ymax
53 66.67        Y increment
54 STO 43
55 5            No. functions
56 XEQ "SHP"    Call to SHP
57 END
```



```
| X4                      | X4                      | X4
: X3                      : X3                      : X3
. -50                     . -50.000                 . -50.000
. 0                       . 0.000                   . 0.000
. 50                      . 50.000                  . 50.000
Y: -100.000 TO -33.333    Y: -33.333 TO 33.333      Y: 33.333 TO 100.000
X: -5.000 TO 5.000        X: -5.000 TO 5.000        X: -5.000 TO 5.000
ΔX=0.020                  ΔX=0.020                  ΔX=0.020
```

```
106 FS? 01
107 " *"          Add asterisk
108 FS?C 01
109 ACA
110 PRBUF
111 END
```

We initialize by clearing registers 6 through 22 with the BC routine, and load in the input to HS . Then, the user is prompted 'READY' for test scores. After all scores have been entered, the histogram is printed, along with the mean and standard deviation:

Keystrokes	Display	Result
XEQ 'TSTPLT	READY	Initializes registers, clears R06-R22
1st score, XEQ A	1.0000	First score in, prints value
2nd score, XEQ A	2.0000	2nd score in, printed
.	.	.
.	.	.
.	.	.
Nth score, XEQ A	N.0000	Last score in, printed
XEQ B		Prints histogram, mean, and standard deviation
To begin again, XEQ a	READY	Initializes, etc.
1st score, XEQ A	1.0000	First score in, prints value

After all the scores have been entered and printed, the histogram in figure 3 is produced.

```
 1 = 70.0000
 2 = 80.0000
 3 = 32.0000
 4 = 75.0000
 5 = 76.0000
 6 = 89.0000
 7 = 95.0000
 8 = 62.0000
 9 = 100.0000
10 = 79.0000
11 = 74.0000
12 = 81.0000
13 = 79.0000
14 = 77.0000
15 = 73.0000
16 = 51.0000
17 = 76.0000
18 = 65.0000
19 = 78.0000
20 = 74.0000
```

```
5% PER DIAMOND
( MAX. = 50% )
 0-9
 10-19
 20-29
 30-39 -
 40-49
 50-59 -
 60-69 ♦-
 70-79 ♦♦♦♦♦♦♦♦♦♦♦ *
 80-89 ♦♦-
 90-99 -
 100   -
 * = > 50%

MEAN = 74.0500

S.D. = 14.4093
```

Figure 3. A histogram of class grades entered into program TSTPLT from table 3 above.

The original goal of the histogram plot here was to have each single diamond character in a bar represent 5 percent of the total value of the student population. Thus, if the maximum height of a column could represent 50 percent, then a 70 column maximum height would assure 5 percent per fill character. However, because the last full 7 columns would be made up of fill columns since the tenth diamond wouldn't quite reach the 70th position, this goal couldn't be met. (See the limitation discussion above.) In order to assure a 10 diamond column for a full column, the plot width was made to be 71 columns. Then, HS would fill it with ten complete diamond characters plus a single additional fill column of ACCOL 8.

This program was submitted by Jack Sutton (5622) during the documentation phase of the PPC ROM project.

FURTHER DISCUSSION OF HS

Vertical Character Accumulation. This routine, originally submitted for inclusion in the PPC ROM, was written by Cliff Carrie (834). It is extremely useful for labelling the X direction of histograms, bar charts or any plots on the 82143A printer. Merely key in a number between 0 and 99 inclusive and the 2 digits will be accumulated into the print buffer as 5 ACCOL columns. If flag 12 is set when ACV is called, then the digits become twice as tall. The routine ACV listing:

BAR CODE ON PAGE 479

```
APPLICATION PROGRAM FOR: HS

01♦LBL "ACV"
02 10
03 /              Separate into
04 ENTER↑         first and second
05 FRC            digits
06 10
07 *
08 XEQ IND Y      Get 1st digit code
09 XEQ IND Y      Get 2nd digit code
```

```
10♦LBL 14
11 10
12 *
13 FRC
14 LASTX          Combine two codes
15 INT            to create ACCOL
16 16             values and accum-
17 *              ulate them into
18 RCL Z          print buffer
19 10
20 *
21 +
22 ACCOL
23 FRC
24 X<>Y
25 X≠0?
26 GTO 14
27 RTN            Codes for digits:
28♦LBL 00        0
29 .25552
30 RTN
31♦LBL 01        1
32 .22232
33 RTN
34♦LBL 02        2
35 .72452
36 RTN
37♦LBL 03        3
38 .34243
39 RTN
40♦LBL 04        4
41 .47564
42 RTN
43♦LBL 05        5
44 .34317
45 RTN
46♦LBL 06        6
47 .25316
48 RTN
49♦LBL 07        7
50 .22247
51 RTN
52♦LBL 08        8
53 .25252
54 RTN
55♦LBL 09        9
56 .34652
57 END
```

The barcode for routine ACV appears in Appendix N.

Example 4. Plot the following function: Y = EXP(-(X^2)/100) using HS . Let X values range from 0 to 20, in increments of 1. Label the columns using routine ACV. Y limits shall be from 0 to 1 inclusive.

Since ACV only occupies 5 printer columns, let us use 155 for the plot width and print the X labels double width (10 colums). We can choose printer symbol number 127 for a fill character and ACCOL number 127 for a fill column:

APPLICATION PROGRAM FOR: **HS**

```
01♦LBL "PLOT"
02 .02
03 STO 00        Numeric counter
04 155
05 STO 04        Plot width
06 127
07 STO 03        Fill character
08 STO 05        Fill column
09♦LBL 00
10 RCL 00        X label value
11 SF 12         Set double width
12 XEQ "ACV"     Accumulate label
13 CF 12
14 2
15 SKPCOL
16 RCL 00        Compute Y height
17 X↑2           of bar
18 1 E2
19 /
20 CHS
21 E↑X
22 XROM "HS"     Call HS
23 PRBUF         Print buffer
24 ISG 00
25 GTO 00
26 END
```

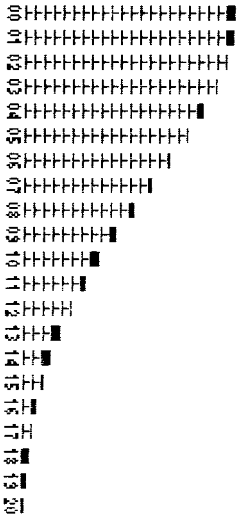


Figure 4. Plot of the function in Example 4, using the ACV routine to accumulate X labels, and using HS to produce histogram bars.

Routine Listing For: **HS**

```
48♦LBL "HS"
49 RCL 04
50 *
51 LASTX
52 X>Y?
53 X<>Y
54 INT
55 7 E-5
56 +
57 RCL 03
58 GTO 00
59♦LBL 01
60 ACCHR
61♦LBL 00
62 DSE Y
63 GTO 01
64 RDN
65 INT
66 8
67 +
68 RCL 05
69 GTO 00
70♦LBL 02
71 ACCOL
72♦LBL 00
73 DSE Y
74 GTO 02
75 RTN
```

LR - LENGTHEN RETURN STACK

The 41C operating system provides for six levels of subroutine calls by storing the six return addresses in status registers a and b. If more pending returns are needed, existing returns can be stored in a data register pair by the LR routine, freeing the status registers to held six more addresses. However, there can be a maximum of five return addresses pending when LR is called, since the instruction XEQ LR uses the sixth return address. The old return addresses can be restored by the SR routine.

Example 1: Suppose that you wish to call a hypothetical ROM routine XX, which is known to use three subroutine levels from the fourth subroutine level of your program ABC. This requires seven subroutine levels and normally would not be possible on the 41C. However, by using a single call to LR (and to SR) up to eleven levels may be used. The following program uses registers 11 and 12 to store the return stack.

```
LBL "ABC"
   :
LBL 01     LBL 01 called at 4th subroutine level
   .
11
XROM LR    Saves return stack in registers 11 & 12
XROM XX    Can freely use up to six subroutine levels
11
XROM SR    Restores original return stack
   :
END
```

COMPLETE INSTRUCTIONS FOR LR

LR will store up to five subroutine return addresses in a data register pair selected by the user. The routine does not alter the contents of status registers a or b. The user's program must put the number of the first register of the pair in the X register before LR is called. The Y, Z, and T registers are returned in X, Y, and Z after execution, and LastX and all ALPHA registers are lost.

SR recalls five return addresses from a data register pair and stores them in status registers a and b. The information in the data register pair is not altered by SR, and may be used again if desired. The number of the first register of the pair must be in X when SR is called, and Y, Z, and T are returned in X, Y, and Z after execution. LastX and ALPHA are lost.

MORE EXAMPLES OF LR

Example 2: The SUB1 routine is a demonstration of extended subroutine stack depth. The user places in X the desired subroutine depth, which can be up to 770 levels, depending on the amount of available memory. The formula is max levels = 5*[INT (SIZE/2) +1]. When the routine is run, it executes repeated subroutine calls, displaying the current subroutine level, until the desired depth has been reached, when it beeps and starts executing repeated returns, counting back down until all subroutines have been returned from. This program was written by Keith Jarett (4360) as a test routine for LR and SR during the ROM loading process.

BAR CODE ON PAGE 483

APPLICATION PROGRAM FOR: LR

```
01♦LBL "SUB1"      31♦LBL 14
02  E3             32 R↑
03 /               33 R↑
04♦LBL 01          34 XEQ 01
05 VIEW X          35 RCL X
06 ISG X           36  E
07 GTO 14          37 X=Y?
08 BEEP            38 GTO 14
09 INT             39 -
10 DSE X           40 5
11 RTN             41 XROM "QR"
12♦LBL 14          42 X≠0?
13 RCL X           43 GTO 14
14 INT             44 RDN
15  E              45  E
16 X=Y?            46 -
17 GTO 14          47 2
18 -               48 *
19 5               49 XROM "SR"
20 XROM "QR"       50 R↑
21 X≠0?            51 R↑
22 GTO 14          52♦LBL 14
23 RDN             53 R↑
24  E              54 R↑
25 -               55 VIEW X
26 2               56 DSE X
27 *               57 RTN
28 XROM "LR"       58 PSE
29 R↑              59 VIEW X
30 R↑              60 BEEP
                   61 .END.
```

APPLICATION PROGRAM 1 FOR

LR and SR are simple to use when the depth of subroutine calls is a constant. However, for recursive algorithms the program determines the depth of subroutine usage, and managing the return stack becomes more difficult. The two programs LRR (lengthen return stack for recursion) and SRR (shorten return stack for recursion) provide automatic management of the return stack by calling LR and SR only when needed. These routines require two data registers for level counting, two registers for each use of LR, and a short initialization (IRX) before the routines can be used. LRR and SRR automatically allow for curtain moving, which is usually needed to support recursion. They use the top 2+2k data registers, where k represents the maximum number of times LR is called. Since LR is called for each 5 subroutine levels this means that 2+2*INT(n/5) registers are used, where n is the maximum number of subroutine levels. The top data register is used by LRR and SRR as a subroutine level counter, while the penultimate register contains a pointer of the form iii.fff02 used to access registers for LR and SR, with iii ≥ fff.

The return stack management routines are used as follows:

1) Make sure your SIZE is sufficient for what you want to do. Then place in X the number of lowest register to be used for the extended return stack. (The return stack is actually constructed from high registers to low registers, but this number will provide a lower bound to protect other data that you may need. If you don't need this protection use 1 or 0.) XEQ "IRX" (initialize for recursive execution) to initialize the top two registers for LRR and SRR.

2) After each LBL which initiates a chain of calls more than two deep (this includes all recursive labels, but does not include utility routines which themselves call only one level) you must XEQ "LRR". The XEQ "LRR" may be anywhere between the LBL and the first XEQ instruction, but the recommended location is immediately after the LBL.

3) Likewise, before a RTN is executed from a program segment that initiates a chain of calls more than two deep you must XEQ "SRR". The XEQ "SRR" may be anywhere between the last XEQ instruction and the RTN, but the recommended location is immediately before the RTN. It is also recommended that all return paths be funnelled to a single RTN instruction , so that only one XEQ "SRR" is required.

4) Because of the nature of LRR and SRR it is always possible to call a two level subroutine without using LRR and SRR. In this way, utility routines which themselves call at most one other subroutine may be used efficiently. An example of this technique is the use of PUSH and POP in Example 4.

5) Only two parameters in the stack (X and Y registers) remain intact after execution of LRR or SRR. However, you may insert any number of steps between the LRR call and its associated RTN. In this way, the stack and ALPHA may be emptied or filled as required.

6) The routines are shown here with global labels for clarity; however, if possible they should be used with local labels to allow the XEQ branches to be compiled. This will increase execution speed as well as reduce the byte count.

Example 3: The SUB2 routine operates identically to the SUB1 routine, except for some unavoidable display scrolling (see IF Example 6), but it has been modified to use LRR and SRR. The modified version is more compact and is easier to understand, since return stack management is not performed by the routine itself. However, the SUB1 routine is more efficient because it does not use data registers (all indexing is done in the stack) and it is shorter because it only calls PPC ROM routines, whereas LRR, SRR, and IRX must all be in memory for SUB2 to operate. There is an interesting tradeoff, though, because a routine such as SUB2 can be written and debugged in a much shorter time. Unless the maximum capacity of the 41C is needed, it is probably not worth the required programming effort to make your routine perform its own return stack management.

BAR CODE ON PAGE 484

APPLICATION PROGRAM FOR: **LR**

```
01♦LBL "SUB2"
 02  E3
 03 /
 04  E
05 XEQ "IRX"

 06♦LBL 01
07 XEQ "LRR"
 08 VIEW X
 09 ISG X
 10 GTO 02
 11 TONE 9
 12 INT
 13 GTO 03

 14♦LBL 02
 15 XEQ 01
 16 VIEW X

 17♦LBL 03
18 XEQ "SRR"
 19 DSE X
 20 RTN
 21 PSE
 22 TONE 5
 23 CLD
 24 END
```

BAR CODE ON PAGE 484

APPLICATION PROGRAM FOR: **LR**

```
01♦LBL "IRX"
 02 CHS
 03 .02
 04 +
05 XROM "S?"
 06  E
 07 -
 08 .
09 STO IND Y
 10 RDN
 11 +
 12  E3
 13 ST/ Y
 14 RDN
 15 DSE L
16 STO IND L
 17 RDN
 18 RTN

19♦LBL "LRR"
 20 SF 14
 21 GTO 00

22♦LBL "SRR"
 23 CF 14

 24♦LBL 00
 25 RCL c
 26 STO [
 27 "├♦♦♦♦"
 28 X<> [
 29 X<> d
 30 CF 02
 31 CF 03
 32 X<> d
 33 ENTER↑
 34 INT
 35 HMS
 36 X<>Y
 37 SIGN
 38 RDN
 39 7
 40 ST* Y
 41 X<> L
 42 +
 43 - E1
 44 *
 45 INT
 46 64
 47 MOD
 48 SF 25

 49♦LBL 01
50 RCL IND X
 51 FC? 25
 52 GTO 02
 53 X<> L
 54 +
 55 GTO 01

 56♦LBL 02
 57  E
 58 -
 59 STO [
 60 X<> L
 61 FC?C 14
 62 GTO 05
63 X<> IND L
64 ST+ IND L
 65 X=0?
 66 GTO 04
 67 5
 68 MOD
 69 X≠0?
 70 GTO 04
 71 DSE [
 72 RDN
73 ISG IND [
 74 FS? 53
 75 GTO 03
76 RCL IND [
 77 INT
 78 ST- [
 79 X<> [
 80 GTO "LR"

 81♦LBL 03
82 "NO ROOM- LRR"
 83 PROMPT

 84♦LBL 04
 85 RDN
 86 CLA
 87 RTN

 88♦LBL 05
89 ST- IND L
 90 RDN
91 RCL IND L
 92 X=0?
 93 GTO 04
 94 5
 95 MOD
 96 X≠0?
 97 GTO 04
 98 DSE [
 99 RDN
100 RCL IND [
 101 INT
 102 X=0?
 103 GTO 06
104 DSE IND [
 105 ""
 106 ST- [
 107 X<> [
 108 GTO "SR"

 109♦LBL 06
110 "TOO FAR-SRR"
 111 PROMPT
 112 END
```

The LRR and SRR routines are especially useful for implementing recursive algorithms on the 41C. If your algorithm is not recursive, the direct method of Example 1 may be better. There are many problems that lend themselves to recursive solutions--one of the simplest of these is the computation of a factorial. With a high level computer language that supports recursion, a factorial algorithm could be implemented in the following two statements:

```
PROCEDURE FACT(N)
FACT    = 1
IF N = 1 THEN RETURN
ELSE FACT    = N * FACT(N-1)
```

If the 41C had a large enough operational stack and return stack, a factorial routine could be written as follows:

```
01 LBL "FCT"
02 ENTER↑
03 DSE X
04 XEQ "FCT"
05 X=0?
06 SIGN
07 *
08 RTN
```

The zero test and SIGN merely prevent multiplication by zero. This routine correctly calculates the factorial of 1, 2, or 3 but fails for larger numbers, because all four stack registers are used. By using the PUSH and POP routines (and the IRX initialization routine) to form an indefinitely long stack in memory, and the LRR and SRR routines to provide an extended return stack, a recursive factorial routine can easily be written for the HP-41.

Example 4: This program is given for illustrative purposes only. Because of its simplicity, it is a good example of recursive programming techniques; however, it obviously has no use as a computational tool since the 41C FACT function is hundreds of times faster and uses no data registers.

Both the return stack management routines and the 'infinite' stack routines need initialization before FCT can be run. To evaluate up through 69!, execute the following:

SIZE $\geq$ 98 (= n + 3 + 2 * INT (n/5))

XEQ "IRX"

To evaluate factorials, just enter an integer between 1 and 69 and XEQ "FCT". The registers do not have to be re-initialized unless the program is stopped before completion or if register 00 or the highest two data registers are altered.

APPLICATION PROGRAM FOR: **LR**

```
01♦LBL "FCT"
02 XEQ "LRR"
03 XEQ "PUSH"
04 DSE X
05 GTO 21
06 X=0?
07 SIGN
08 GTO 22

09♦LBL 21
10 XEQ "FCT"

11♦LBL 22
12 XEQ "POP"
13 *
14 XEQ "SRR"
15 RTN

16♦LBL "PUSH"
17 STO IND 00
18 ISG 00
19 ""
20 RTN

21♦LBL "POP"
22 DSE 00
23 ""
24 RCL IND 00
25 RTN

26♦LBL "INIT"
27  E
28 STO 00
29 XEQ "IRX"
30 END
```

While the factorial program has a simpler non-recursive solution on the 41C, there are many routines that are extremely difficult to solve unless recursive methods are used. An example of this is the Towers of Hanoi program by Harry Bertuccelli (3994), covered elsewhere in this manual.

APPLICATION PROGRAM 2 FOR LR

If your recursive program calls itself from only one point, then the return addresses stored by LR are redundant. This means that there is probably a simple nonrecursive looping solution to your problem, but if you want to use recursion you need not pay LRR's heavy penalty in register usage.

The LRS (lengthen return stack with single return address) and SRS (shorten return stack with single return address) supportive routines are similar to LRR and SRR with the following exceptions. LR is called only once, at the fifth level. LRS assumes that all return addresses are identical. SRS calls SR every five levels as does SRR, but it always places the same return pointers in status registers a and b. Only the top three data registers are used. No input is required for ISX (initialize for single return address execution).

BAR CODE ON PAGE 484

APPLICATION PROGRAM FOR: **LR**

```
01♦LBL "ISX"
02 XROM "S?"
03 DSE X
04 .
05 STO IND Y
06 R↑
07 R↑
08 RTN

09♦LBL "LRS"
10 SF 14
11 GTO 00

12♦LBL "SRS"
13 CF 14

14♦LBL 00
15 RCL c
16 STO [
17 "⊦♦♦♦♦"
18 X<> [
19 X<> d
20 CF 02
21 CF 03
22 X<> d
23 ENTER↑
24 INT
25 HMS
26 X<>Y
27 SIGN
28 RDN
29 7
30 ST* Y
31 X<> L
32 +
33 - E1
34 *
35 INT
36 64
37 MOD
38 SF 25
39 CLA

40♦LBL 01
41 RCL IND X
42 FC? 25
43 GTO 02
44 X<> L
45 +
46 GTO 01

47♦LBL 02
48  E
49 -
50 FC?C 14
51 GTO 04
52 X<> L
53 X<> IND L
54 ST+ IND L
55 5
56 X=Y?
57 GTO 03
58 R↑
59 R↑
60 RTN

61♦LBL 03
62 R↑
63 R↑
64 LASTX
65 2
66 -
67 GTO "LR"

68♦LBL 04
69 STO [
70 X<> L
71 ST- IND L
72 RDN
73 RCL IND L
74 X=0?
75 STO IND L
76 X=0?
77 GTO 05
78 5
79 MOD
80 X≠0?
81 GTO 05
82 X<> [
83 CLA
84 2
85 -
86 GTO "SR"

87♦LBL 05
88 RDN
89 END
```

APPLICATION PROGRAM FOR: LR

```
01♦LBL "SUB2"
 02 E3
 03 /
 04 XEQ "ISX"

 05♦LBL 01
 06 VIEW X
 07 ISG X
 08 GTO 02
 09 TONE 9
 10 INT
 11 GTO 03

 12♦LBL 02
 13 XEQ "LRS"
 14 XEQ 01
 15 XEQ "SRS"
 16 VIEW X

 17♦LBL 03
 18 DSE X
 19 RTN
 20 PSE
 21 TONE 5
 22 CLD
 23 END
```

The modified versions of SUB2 and FCT shown here use LRS and SRS. This saves registers and increases speed. Both SUB2 and FCT satisfy the essential constraint that there is only one XEQ instruction that is recursive. Because of this constraint it is simplest to surround the recursive XEQ instruction with XEQ "LRS" above and XEQ "SRS" below.

BAR CODE ON PAGE 484

APPLICATION PROGRAM FOR: LR

```
 01♦LBL "FCT"
 02 XEQ "PUSH"
 03 DSE X
 04 GTO 21
 05 X=0?
 06 SIGN
 07 GTO 22

 08♦LBL 21
 09 XEQ "LRS"
 10 XEQ "FCT"
 11 XEQ "SRS"

 12♦LBL 22
 13 XEQ "POP"
 14 *
 15 RTN

 16♦LBL "PUSH"
 17 STO IND 00
 18 ISG 00
 19 ""
 20 RTN

 21♦LBL "POP"
 22 DSE 00
 23 ""
 24 RCL IND 00
 25 RTN

 26♦LBL "INIT"
 27 E
 28 STO 00
 29 XEQ "ISX"
 30 END
```

LINE BY LINE ANALYSIS OF LR

Status registers a and b contain the program pointer and six return addresses - each of these are two bytes (16) long. In the following analysis, a letter "P" will be used to represent each byte of the program pointer, a digit "1" for each byte of the first return address, a "2" for the second return address and so on. Using this notation, registers a and b have the following configuration:

3	2	2	1	1	P	P	b
6	6	5	5	4	4	3	a

To extend the return stack, only the second through the sixth return addresses must be stored- the first return address provides a return to the program that called LR or SR, and the pointer just contains the absolute address of the program step where register b was recalled. The five return addresses that are

STACK AND ALPHA REGISTER ANALYSIS FOR LR

T	Z	Y	X	L	P							O							N							M							L-# INSTRUCTION
T_0	Z_0	Y_0	n	L																													26♦LBL "LR"
T_0	Z_0	Y_0	1	n																													27 SIGN
1	T_0	Z_0	Y_0	n																													28 RDN
1	T_0	Z_0	Y_0	n																												+	29 "+"
T_0	Z_0	Y_0	6655443	n																												+	30 RCL a
T_0	Z_0	Y_0	6655443	n															6	6	5	5	4	4	3							+	31 STO \
6655443	T_0	Z_0	Y_0	n															6	6	5	5	4	4	3							+	32 RDN
T_0	Z_0	Y_0	32211PP	n															6	6	5	5	4	4	3							+	33 RCL b
T_0	Z_0	Y_0	+	n															6	6	5	5	4	4	3	3	2	2	1	1	P	P	34 X<> [
T_0	Z_0	Y_0	+	n														+	6	6	5	5	4	4	3	3	2	2	1	1	P	P	35 STO]
T_0	Z_0	Y_0	+	n														+	6	6	5	5	4	4	3	3	2	2	1	1	P	P	36 ASTO IND L
T_0	Z_0	Y_0	+	n+1														+	6	6	5	5	4	4	3	3	2	2	1	1	P	P	37 ISG L
T_0	Z_0	Y_0	+	n+1														+	6	6	5	5	4	4	3	3	2	2	1	1	P	P	38 ""
T_0	Z_0	Y_0	+	n+1									+	6	6	5	5	4	4	3	3	2	2	1	1	P	P	*	*	*	*	*	39 "⊢*****"
T_0	Z_0	Y_0	+	n+1														+	4	3	3	2	2	1	1	P	P	*	*	*	*	*	40 STO]
T_0	Z_0	Y_0	+	n+1														+	4	3	3	2	2	1	1	P	P	*	*	*	*	*	41 ASTO IND L
+	T_0	Z_0	Y_0	n+1														+	4	3	3	2	2	1	1	P	P	*	*	*	*	*	42 RDN
+	T_0	Z_0	Y_0	n+1																													43 CLA
+	T_0	Z_0	Y_0	n+1																													44 RTN

Routine Listing For: LR

```
26♦LBL "LR"
27 SIGN
28 RDN
29 "+"
30 RCL a
31 STO \
32 RDN
33 RCL b
34 X<> [
35 STO ]

36 ASTO IND L
37 ISG L
38 ""
39 "⊢*****"
40 STO ]
41 ASTO IND L
42 RDN
43 CLA
44 RTN
```

stored by LR constitute ten bytes, five of which are stored in each register of the pair. Since some of the bytes may be nulls (hex 00), and they are stored directly from the ALPHA register using ASTO (which will store six characters, but skips over leading nulls), an alpha character delimiter must be used to force ASTO to take the correct five bytes. This delimiter is the character "+", which is the sixth byte stored in each register. After execution of LR, the register pair contains the following information stored as alpha strings:

```
Reg n    :  "+66554"
Reg n+ 1 :  "+43322"
```

The SR routine expects to find the return addresses store in this form and merges these addresses onto the current program pointer and first return address and then stores the results into registers a and b.

The majority of both routines consists of ALPHA register shifting--to analyze this in detail, it is probably easiest to use a STACK/ALPHA analysis sheet such as the one in *PPC TECHNICAL NOTES*, V1N3P38. Analysis sheets for LR and SR are printed in this manual. A blank analysis sheet may be found on page 259 of the manual.

CONTRIBUTORS HISTORY FOR LR

Harry Bertuccelli (3994) wrote the first subroutine level extension routines (see *PPC CALCULATOR JOURNAL*, V7N6P8). Paul Lind (6157) completely rewrote LR and SR, and Roger Hill (4940) independently wrote virtually identical routines.

The application programs were written by Harry Bertuccelli and Keith Jarett (4360) based on Paul Lind's idea.

FURTHER ASSISTANCE ON LR

Call Paul Lind (6157) at (206) 525-1033.
Call Harry Bertuccelli (3994) at (213) 846-6390.

NOTES

TECHNICAL DETAILS

XROM: 20,02 — LR — SIZE: 002

Stack Usage:
0 T: USED
1 Z: PREVIOUS T
2 Y: PREVIOUS Z
3 X: PREVIOUS Y
4 L: X + 1

Alpha Register Usage:
5 M:
6 N:
7 O: ALL CLEARED
8 P:

Other Status Registers:
9 Q: NOT USED
10 ⊦: NOT USED
11 a: NOT USED
12 b: NOT USED
13 c: NOT USED
14 d: NOT USED
15 e: NOT USED

ΣREG: UNCHANGED

Data Registers:
R00: TWO CONSECUTIVE REGISTERS SPECIFIED BY THE USER ARE
R06: ALTERED
R07:
R08:

Flag Usage: NONE USED
04:
05:
06:
07:
08:
09:
10:

25:

Display Mode: UNCHANGED

Angular Mode: UNCHANGED

Unused Subroutine Levels:
5 CALLED BY A PROGRAM
6 FROM THE KEYBOARD

Global Labels Called:

Direct	Secondary
NONE	NONE

Local Labels In This Routine:
NONE

Execution Time: .7 seconds.

Peripherals Required: NONE

Interruptible? YES
Execute Anytime? NO
Program File: SR
Bytes In RAM: 40
Registers To Copy: 40

Other Comments:

R22: Robert	R40: Joe
R23: Jeffer	R41: Robins
R24: son	R42: on
R25: 261.2347	R43: 756.4438
R26: Fresno	R44: Peoria
R27: CA	R45: IL

To further exchange Mary Adams and James Masterson (records 1 and 5) key 1 ENTER↑ 5 and XEQ "M1". Then check the data registers to see that the proper exchange has been made.

APPLICATION PROGRAM 1 FOR M1

Matrix Support Routines RRM AND MIO

The routines called RRM and MIO are provided as matrix support routines. RRM calls the ROM routines M1, M2, M3, M4, M5, and BX. MIO calls M4 and M5.

The program titled RRM will transform a matrix into row reduced echelon form. This means the program will calculate determinants and inverses and will solve systems of equations. The RRM program is only 70 lines long (104 bytes), and handles the three matrix problems, either individually or simultaneously, and uses the technique known as partial pivoting which helps reduce round-off error. Moreover, the only limitation on the size of the matrices is the number of available data registers. The RRM program can even be applied to more than one matrix in data memory. If more than 319 registers ever become available for the HP-41C the RRM program may be run without any modifications to handle any size matrix.

Given our present limitation of 319 registers RRM can be used to compute the determinant of a 17x17 matrix, to solve a system of up to 16 linear equations in 16 unknowns, or compute the inverse of a 12x12 matrix. To solve any of these problems simply load the appropriate matrix in the 41C and XEQ "RRM". The desired result, whether it be a determinant or an inverse or the solution to a system of equations will be calculated and left in the 41C.

The second program called MIO is to be used for matrix input/output operations that will automatically store or recall the entries of a matrix consistent with the requirements of the ROM matrix routines M1 - M5. Although RRM and MIO can be merged into one program, the reason for writing two separate matrix modules is to handle as large a matrix as possible. RRM does all the hard work; MIO is only an example of an input/output scheme.

It should be pointed out that it is possible to use other methods to solve the same matrix problems, but for completely automatic operation as RRM and MIO provide we have approached the theoretical limit. If you plan on writing your own matrix routines that will call M1 - M5 the following suggestions will be helpful.

1. M1 - M5 require the starting register of the matrix to be stored in R07 and the number of columns in the matrix be stored in R08. Matrices are stored row by row with each row occupying a block of consecutive registers. The entire matrix is stored as one large block of consecutive registers.

2. Although M1 - M5 do not require the number of rows, the number of rows, if used, should be stored in R09. Later you may find matrix uses for IR and DR which do use R09.

3. To achieve maximum size start storing the matrix entries in R10 on up and use registers R06 and below for program scratch area.

4. Given the above 3 constraints consider further the storage requirements. For the determinant problem, to store a 17x17 matrix requires 289 registers. For the systems of equations problem to hold a 16x16 matrix and one extra column for the constants requires 16x17= 272 registers. For the inverse, unless you are calculating the inverse in place, you will need to store two matrices, one being the original and the other being a form of the identity matrix. Since 12x12 times 2 = 144x2 = 288, you will need 288 registers for the inverse of a 12x12 matrix.

Thus you will need a maximum of 289 registers to solve all 3 types of problems. If you are using R00-R09 for the registers your program requires then 299 registers will already be accounted for before you even start to enter your program. Since 319 - 299 = 20, your program may use approximately 20x7 = 140 bytes.

Both RRM and MIO have been restricted to use less than 140 bytes; RRM is 104 bytes and MIO is 129 bytes. If you are not particular about the maximum capacity available you can combine these two programs and add many of your own bells and whistles to MIO and still have enough data memory available to perform some fancy operations on 10x10 matrices. If RRM is used alone you will have 295 registers available for matrix data. If MIO is used alone 291 data registers will be available for matrix data. When RRM and MIO are combined 276 registers are available.

Three examples follow which illustrate the use of RRM and MIO. To run the examples, first perform "MEMORY LOST" and then SIZE 031 (minimum). Read in the MIO program first and then GTO .. Next read in the RRM program and GTO .. again. Then key CAT 1 and immediately press R/S so that you are in the MIO program. Switching to USER mode makes the following functions available on keys A, B, C.

New Matrix	Review Matrix	Recall (Y,X)

Problem 1: Solve the system of equations:

$$-5X + 10Y + 15Z = 5$$

$$2X + Y + Z = 6$$

$$X + 3Y - 2Z = 13$$

Perform the following operations.

Press	Function	See in Display
A	Initialization for a new matrix	"START REG. ?"
10 R/S	Start storing matrix in R10 and above	"DIM: R?↑C?"
3 ENTER↑ 4 R/S	Key in dimension as 3 rows and 4 columns.	-TONE 9- "(1,1)=?"

We next enter one by one the entries of the coefficient matrix starting with the first row.

$$\left[\begin{array}{ccc|c} -5 & 10 & 15 & 5 \\ 2 & 1 & 1 & 6 \\ 1 & 3 & -2 & 13 \end{array}\right]$$

The program will sound TONE 9 when it is ready for the next entry and will prompt with "(row,column)=?" where row and column are numbers. Key in the next coefficient followed by R/S. For example, the display should still show "(1,1)=?" and the first row would be keyed in as:

See in Display	Press
"(1,1)=?"	5 CHS R/S
"(1,2)=?"	10 R/S
"(1,3)=?"	15 R/S
"(1,4)=?"	5 R/S
"(2,1)=?"	

Continue keying in the 2nd and 3rd rows. After keying in 13 and pressing R/S for the last (3,4) entry the program will sound BEEP to indicate you should be finished entering the data.

You may now verify the data input by pressing B. First however, store a number (say 4) in R05 for the number of decimal places to be displayed. Pressing B will automatically run through the entire matrix. If a printer is connected and turned on key B will give a printout of the entire matrix. If you prefer scientific notation change line 43 in the MIO program from FIX IND 05 to SCI IND 05. A BEEP will sound when the output is finished.

You may also inspect any particular element using key C. Key in the row and column numbers of the matrix element you wish to view and press C. For example, to verify that the (3,2) element is 3, key 3 ENTER↑ 2 and press C. You should first see "R19.0000" and then "(3,2)=3.0000". The indication here is that the (3,2) element is stored in register R19 and is equal to 3.

Note: If you make an incorrect entry during the automatic input phase simply continue entering elements as directed by the display. After all entries have been made you can use key C to make corrections, since pressing C tells you in which register you should manually store the element in question.

To solve the above system simply XEQ "RRM". This first example will run in about 34 seconds. When the program ends key CAT 1 R/S to insure you are in the MIO program and then press B in USER mode to display the final matrix which is:

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array}\right]$$

The solution is X=2, Y=3, and Z=-1. The determinant of the square coefficient matrix is stored in R01. det. = 150.0000

Problem 2: Find the inverse of the matrix:

$$\begin{bmatrix} 2 & -3 & 1 \\ 3 & 2 & -1 \\ 5 & -2 & 1 \end{bmatrix}$$

To use RRM to find the inverse of a square matrix we form the auxiliary matrix which consists of the original matrix augmented by an identity matrix of the same size. For this problem we will input the 3x6 matrix:

$$\left[\begin{array}{ccc|ccc} 2 & -3 & 1 & 1 & 0 & 0 \\ 3 & 2 & -1 & 0 & 1 & 0 \\ 5 & -2 & 1 & 0 & 0 & 1 \end{array}\right]$$

Press	Function	See in Display
A	Initialization for a new matrix	"START REG. ?"
10 R/S	Start storing matrix in R10 and beyond	"DIM: R?↑C?"
3 ENTER↑ 6 R/S	Key in size as 3 rows and 6 columns.	-TONE 9- "(1,1)=?"

Now continue as in the first example and enter the matrix starting with the first row. Then XEQ "RRM". The program will finish in about 45 seconds. Key CAT 1 R/S when the program finishes and press B to display the result:

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 0.1250 & 0.1250 \\ 0 & 1 & 0 & -1 & -0.3750 & 0.6250 \\ 0 & 0 & 1 & -2 & -1.375 & 1.6250 \end{array}\right]$$

The right hand 3x3 matrix is the inverse of the original matrix. The determinant of the original matrix is found by recalling R01. det. = 8.0000

Problem 3: Use RRM to simultaneously solve the following system of equations, find the inverse of the coefficient matrix, and find the determinant of the coefficient matrix.

$$14X + 2Y - 6Z = 9$$

$$-4X + Y + 9Z = 3$$

$$6X - 4Y + 3Z = -4$$

The matrix to be entered will consist of the original coefficient matrix augmented by the identity matrix and augmented by the final column of constants. This is a 3x7 matrix.

$$\left[\begin{array}{ccc|ccc|c} 14 & 2 & -6 & 1 & 0 & 0 & 9 \\ -4 & 1 & 9 & 0 & 1 & 0 & 3 \\ 6 & -4 & 3 & 0 & 0 & 1 & -4 \end{array}\right]$$

Press	Function	See in Display
A	Initialization for a new matrix	"START REG. ?"
10 R/S	Start storing matrix in R10 and beyond	"DIM: R?↑C?"
3 ENTER↑ 7 R/S	Key in dimension as 3 rows and 7 columns	-TONE 9- "(1,1)=?"

Next enter the rows of the above matrix as directed by the display. Then simply XEQ "RRM". When the program ends (about 49 seconds) key CAT 1 R/S. Then press B to display the matrix:

$$\left[\begin{array}{ccc|ccc|c} 1 & 0 & 0 & 0.0631 & 0.0291 & 0.0388 & 0.5000 \\ 0 & 1 & 0 & -0.1068 & 0.1262 & -0.1650 & 2.0000 \\ 0 & 0 & 1 & 0.0162 & 0.1100 & 0.0356 & 0.3333 \end{array}\right]$$

The inverse of the original matrix is the 3x3 matrix in the middle. The ROM routine DF can be used to convert these decimals to fractions. For the mathematical purist who then wishes to see the exact inverse:

$$\begin{bmatrix} 13/206 & 3/103 & 4/103 \\ 11/103 & 13/103 & -17/103 \\ 5/309 & 34/309 & 11/309 \end{bmatrix}$$

The last column contains the solutions of the system of equations and would be interpreted as X=1/2, Y=2, Z=1/3. The determinant of the coefficient matrix can be recalled from R01. det. = 618.

Some final comments about RRM are in order. If you are only interested in the determinant of a matrix then a square matrix is all that RRM requires. In this case the matrix need not be augmented by any extra columns. RRM always leaves the determinant in R01 but this can be changed to any register by changing lines 06, 34, 50, and 52 in the RRM listing.

Systems of equations are solved as in problem 1, inverses are solved as in problem 2, and the combination of inverse and a system of equations is solved as in problem 3. RRM is just as useful for systems of equations which do not have unique solutions. If the determinant in R01 is 0 (or is so small as to be considered 0) then the system of equations may have no solutions or an infinite number of solutions. Since RRM returns the row reduced echelon form, the final matrix will always be row equivalent to the original. The final matrix may then be used to tell immediately where parameters should be inserted and any and all solutions may then be immediately determined. The coefficient matrix need not be square for RRM to operate on it.

Line By Line Analysis of RRM:

Lines 02-07 initialize the program by storing a 1 in R01 for the determinant and setting flag F10 for the BX routine.

Lines 08-12 make R03 & R04 point to the next pivot position.

Lines 13-20 determine when the program ends by checking if either a row or column boundary has been exceeded.

Lines 21-31 set up the block control word for the BX routine.

Lines 32-36 find the pivot number and check if all the remaining column entries are zero in which case the determinant must be zero and only the next column is incremented by branching to LBL 06.

Lines 37-43 make a 1 in the row containing the pivot number.

Lines 44-48 check if the pivot number is already in the pivot position. Lines 049-052 perform a row interchange to move the pivot to the true pivot position and adjust the sign of the determinant accordingly.

Lines 53-70 make 0's in the current pivot column in all rows except the pivot row.

BAR CODE ON PAGE 480

APPLICATION PROGRAM FOR: **M1**

```
01•LBL "RRM"
02 .
03 STO 03
04 STO 04
05 SIGN
06 STO 01
07 SF 10
08•LBL 05
09 ISG 03
10•LBL 06
11 ISG 04
12 ""
13 RCL 08
14 RCL 04
15 X>Y?
16 RTN
17 RCL 09
18 RCL 03
19 X>Y?
20 RTN
21 RCL 04
22 XROM "M5"
23 X<> Z
24 XROM "M5"
25  E3
26 /
27 +
28 RCL 08
29  E5
30 /
31 +
32 XROM "BX"
33 RCL IND [
34 ST* 01
35 X=0?
36 GTO 06
37 1/X
38 RCL [
39 INT
40 XROM "M4"
41 RDN
42 STO 02
43 XROM "M2"
44 RCL 02
45 ST- 02
46 RCL 03
47 X=Y?
48 GTO 07
49 XROM "M1"
50 RCL 01
51 CHS
52 STO 01
53•LBL 07
54 ISG 02
55 ""
56 RCL 09
57 RCL 02
58 X>Y?
59 GTO 05
60 RCL 03
61 X=Y?
62 GTO 07
63 RCL 02
64 RCL 04
65 XROM "M5"
66 RDN
67 RCL IND T
68 CHS
69 XROM "M3"
70 GTO 07
```

BAR CODE ON PAGE 480

APPLICATION PROGRAM FOR: **M1**

```
01♦LBL "MIO"
02♦LBL A
03 "START REG. ?"
04 AVIEW
05 STOP
06 STO 07
07 "DIM: R?↑C?"
08 AVIEW
09 STOP
10 STO 08
11 X<>Y
12 STO 09
13 SF 09
14 GTO 01
15♦LBL B
16 CF 09
17♦LBL 01
18 CF 29
19 RCL 07
20 STO 04
21 RCL 08
22 RCL 09
23 *
24 STO 03
25♦LBL 02
26 RCL 04
27 XROM "M4"
28 FIX 0
29 " ("
30 ARCL Y
31 "⊢,"
32 ARCL X
33 "⊢)="
34 FC? 09
35 GTO 03
36 "⊢?"
37 AVIEW
38 TONE 9
39 STOP
40 STO IND 04
41 GTO 04
42♦LBL 03
43 FIX IND 05
44 ARCL IND 04
45 AVIEW
46♦LBL 04
47 ISG 04
48 ""
49 DSE 03
50 GTO 02
51 BEEP
52 RTN
53♦LBL C
54 XROM "M5"
55 " R"
56 ARCL X
57 AVIEW
58 STO 04
59  E
60 STO 03
61 CF 09
62 GTO 02
```

Routine Listing For: **M1**

```
28♦LBL "M1"
29 XEQ 00
30 X<>Y
31 XEQ 00
32♦LBL "BE"
33 RCL IND Y
34 X<> IND Y
35 STO IND Z
36 RDN
37 ISG X
38 ""
39 ISG Y
40 GTO "BE"
41 RTN
42♦LBL 00
43 RCL 08
44 *
45 RCL 07
46 +
47 RCL X
48 RCL 08
49 ST- Z
50 SIGN
51 -
52  E3
53 /
54 +
55 RTN
```

LINE BY LINE ANALYSIS OF M1

M1 feeds into the block exchange routine BE after setting up the two block control words for the two rows by calling the local label LBL 00 twice. If R07=s=starting register of the matrix and R08=c=the number of columns in the matrix and i=the row number of the ith row, then with i in X, LBL 00 computes bbb.eee=the block control word for row i.

bbb = s + c*(i-1)   eee = s + c*i - 1

CONTRIBUTORS HISTORY FOR M1

The M1 routine and documentation are by John Kennedy (918).

FURTHER ASSISTANCE ON M1

John Kennedy (918) phone: (213) 472-3110 evenings
Richard Schwartz (2289) phone: (213) 447-6574 eve.

TECHNICAL DETAILS

XROM: 20, 33 **M1** SIZE: depends on matrix size

Stack Usage:
0 T: used
1 Z: used
2 Y: used
3 X: used
4 L: used

Alpha Register Usage:
5 M: not used
6 N: not used
7 O: not used
8 P: not used

Other Status Registers:
9 Q: not used
10 ⊢: not used
11 a: not used
12 b: not used
13 c: not used
14 d: not used
15 e: not used

ΣREG: not used

Data Registers:
R00: not used
R06: not used
R07: s=start reg. matrix
R08: c=# columns in matrix
R09: not used
R10: not used
R11: not used
R12: not used

Flag Usage:
04: not used
05: not used
06: not used
07: not used
08: not used
09: not used
10: not used
25: not used

Display Mode:
not used

Angular Mode:
not used

Unused Subroutine Levels:
4

Global Labels Called:

Direct	Secondary
none falls into BE routine	none

Local Labels In This Routine:
00

Execution Time: depends on matrix size.
1.07C + 0.56 seconds where C = # columns in matrix

Peripherals Required: none

Interruptible? yes

Execute Anytime? no

Program File: M2

Bytes In RAM: 56

Registers To Copy: 61

Other Comments:

```
: S3X
· TIC1
· TIC2
· TIC3
: 0.00
Y: -1.500 TO 1.500
X: 0.000 TO 360.000
ΔX=10.000
```

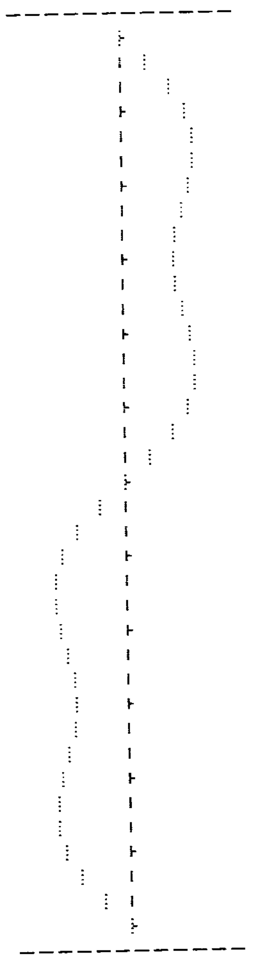


Figure 22. Plot of the function Y = sin X + (1/3)sin 3X from Example 19 with tic marks printed on the axis every 30 degrees. Execution time: 18 min 19 sec.

J.4. Automatic Computation of Y Limits of a Function.

In some cases, a user may wish to plot a function whose behavior is unknown in the desired range of X values. Here is a program written by Jack Sutton (5622) which accepts X inputs and will print the Y minimum and Y maximum of a given function for each 10 values of X between the X limits, inclusive. This program uses the ROM routine **BX** (Block Extremes) to find the Y limits for each 10-value range.

BAR CODE ON PAGE 479

APPLICATION PROGRAM FOR: **MP**

```
01♦LBL "MAXMIN"
02 CF 21
03 .014
04 XROM "BC"
05 "X MIN?, R/S"
06 PROMPT
07 STO 11
08 "X MAX?, R/S"
09 PROMPT
10 STO 12
11 "X INC?, R/S"
12 PROMPT
13 STO 13
14 AON
15 "F(X) NAME"
16 "⊢?, R/S"
17 PROMPT
18 ASTO 14
19 AOFF
20 SF 21
21♦LBL "BLK"
22 "FROM "
23 ARCL 11
24 PRA
25 1.01
26 STO 00
27 XROM "BC"
28♦LBL "CAL"
29 RCL 12
30 RCL 11
31 X>Y?
32 GTO 02
33 XEQ IND 14
34 STO IND 00
35 RCL 11
36 RCL 13
37 +
38 STO 11
39 ISG 00
40 GTO "CAL"
41♦LBL 02
42 "TO "
43 RCL 11
44 RCL 13
45 -
46 ARCL X
47 "⊢ :"
48 PRA
49 RCL 00
50 1
51 -
52 INT
53 1 E3
54 /
55 1
56 +
57 XROM "BX"
58 XEQ 00
59 RCL 12
60 RCL 11
61 X<=Y?
62 GTO "BLK"
63 ADV
64 "DONE"
65 PRA
66 XROM "PO"
67 RTN
68♦LBL 00
69 "MIN="
70 ARCL X
71 PRA
72 "MAX="
73 ARCL Y
74 PRA
75 ADV
76 .END.
```

The barcode for MAXMIN appears in Appendix **N**.

<u>Example 20.</u> Use the MAXMIN program to find the Y limits for the function Y=3*X↑2+6*X+4 between the X values of -10 and 10, with X increment of X=0.2:

First, write the program for the function to be analyzed:

```
01♦LBL "POLY"
02 STO Y
03 X↑2
04 3
05 *
06 X<>Y
07 6
08 *
09 +
10 4
11 +
12 END
```

Now, XEQ MAXMIN and enter the data when prompted:

KEYSTROKES	DISPLAY	RESULT
XEQ MAXMIN	X MIN?, R/S	Prompt for Xmin
10 CHS R/S	X MAX?, R/S	Store Xmin, prompt
CHS R/S	X INC?, R/S	Store Xmax, prompt
.1 R/S	F(X) NAME?, R/S	Store Xinc, prompt
POLY R/S	----	Store Fcn name, then print Ymin,Ymax for each 10 X values

```
FROM -10.000           FROM 2.000
TO -8.200 :            TO 3.800 :
MIN=156.520            MIN=28.000
MAX=244.000            MAX=70.120

FROM -8.000            FROM 4.000
TO -6.200 :            TO 5.800 :
MIN=82.120             MIN=76.000
MAX=148.000            MAX=139.720

FROM -6.000            FROM 6.000
TO -4.200 :            TO 7.800 :
MIN=31.720             MIN=148.000
MAX=76.000             MAX=233.320

FROM -4.000            FROM 8.000
TO -2.200 :            TO 9.800 :
MIN=5.320              MIN=244.000
MAX=28.000             MAX=350.920

FROM -2.000            FROM 10.000
TO -0.200 :            TO 10.000 :
MIN=1.000              MIN=364.000
MAX=4.000              MAX=364.000

FROM 0.000
TO 1.800 :             DONE
MIN=4.000
MAX=24.520
```

Figure 23. Output of the MAXMIN program showing Y limits of the function in Example 20. These values may now be used to select the Ymin and Y max inputs to MP for plotting. Execution time: 3 min 16 sec

J.5. Plots Using Multiple Paper Widths - 'Superplotting'.

When higher plot resolution is desired in the Y direction (across the printer paper) than can be obtained with 168 columns, it is possible to plot graphs with MP which require multiple widths of printer paper. This has been referred to as 'superplotting'. The routine shown below takes care of the housekeeping involved in printing each section of the plot, re-initializes the inputs and increments the Y limits. The only difference between the inputs for this program and for MP is that Ymax is stored in R35 instead of R01, and a Y increment value (the desired width of each printed plot section) is stored in R36. After all the function names are stored, simply set the limits and XEQ SMP:

1. Place the function names in R15 and up
2. Set disappearing overflow mode (CF05,SF06) so functions jump from strip to strip
3. Store Xmin, Xmax and Xinc in R08,R09,& R10
4. Store plot width in R02
5. Store Ymin in R00, Ymax in <u>R35</u> and Yinc in <u>R36</u>
6. Enter the number of functions to be plotted
7. XEQ SMP, and the plot is printed, a strip at a time, moving from Ymin to Ymax in steps equal to the Y increment stored in R36.

The SMP listing is as follows:

```
APPLICATION PROGRAM FOR:  MP

01♦LBL "SMP"      MP superplotting
02 STO 38         Save # fcns in R38
03 RCL 08
04 STO 37         Xmin in R37
05 RCL 00
06 RCL 36
07 +
08 STO 01         Ymin + Yincrement
09♦LBL 00
10 RCL 38         Restore # fcns
11 XROM "MP"      Call to MP
12 RCL 01
13 RCL 35
14 X<=Y?          If done, stop
15 RTN
16 RDN            If not, increment
17 STO 00         Ymin, Ymax
18 RCL 36
19 ST+ 01
20 RCL 37
21 STO 08
22 GTO 00
23♦LBL "SHP"      HP superplotting
24 STO 45         Save # fcns in R45
25 RCL 08
26 STO 44         X min in R44
27 RCL 00
28 RCL 43
29 +
30 STO 01         Ymin + Yincrement
31♦LBL 01
32 RCL 45         Restore # fcns
33 XROM "HP"      Call to HP
34 RCL 01
35 RCL 42
36 X<=Y?          If done, stop
37 RTN
38 RDN            If not, increment
39 STO 00         Ymin, Ymax
40 RCL 43
41 ST+ 01
42 RCL 44
43 STO 08
44 GTO 01
45 END
```

The first plot strip has Ymin=Ymin and Ymax= Ymin+Yinc. The next strip has Ymin= the previous Ymax and Ymax=(new Ymin)+Yinc. This process repeats until the current Ymax exceeds that which was stored into R35. If Yinc is not chosen properly, the last plot strip will exceed the designated upper limit in the Y direction, but the excess may be removed by the user with a scissors if so desired.

NP - NEXT PRIME

This routine will search to find prime factors of an integer n. More specifically, the routine begins its search from a starting trial divisor that the user inputs. NP returns only the next divisor of n. When NP is iterated on itself, starting with 2 as the first trial divisor, all the prime factors of n can be found one by one in increasing order. Intermediate processing can be done between successive prime factors and the routine NP can easily be returned to in order to continue the factorization of n. NP is valid for 10-digit integers n.

Example 1: Find the prime factors of 27,930.

The starting trial divisor will be 2.

Do:	See:	Result:
27930 ENTER↑ 2		enter initial inputs
XEQ "NP"	2	first prime factor
R/S	3	second prime factor
R/S	5	third prime factor
R/S	7	fourth prime factor
R/S	7	fifth prime factor
R/S	19	sixth prime factor
R/S	1	routine finished

The factor just before 1 is returned is the last prime factor. For this example,

27,930 = 2*3*5*7*7*19

Example 2: The number 40,013,933 is known to have only two prime factors, one of which is greater than 5000. Find the two factors of 40,013,933.

We may start with the next odd number greater than 5000.

Do:	See:	Result:
40013933 ENTER↑ 5001		Enter initial inputs
XEQ "NP"	5,309	after 41 seconds
R/S	7,537	second factor
R/S	1	routine finished

The two factors of 40,013,933 are 5,309 and 7,537.

COMPLETE INSTRUCTIONS FOR NP

NP will find the next divisor of an integer n starting from a given trial divisor d which may be 2 or any odd number. The search does not extend beyond the square root of n and if no divisor is found up to that point then n is returned. The divisor the routine returns will be prime provided n has no prime factors strictly smaller than d. Otherwise the divisor returned need not be prime. n may be any 10-digit integer.

1) The integer n may be any positive integer greater than or equal to 1. The trial divisor d must be 2 or an odd integer greater than 2.

2) Key n ENTER↑ d and XEQ "NP".

3) The routine ends with n in Y and p in X where p is a divisor of n. p is also returned in LAST X.

4) If NP is executed from the keyboard, when the next divisor is returned, immediately pressing R/S will cause NP to continue searching for the next factor. The divisor returned may repeat, but when the routine returns 1 there are no more factors of n.

Example 3: Determine whether or not 99,991 is prime.

NP can be used to test potential primes by choosing 2 as the starting trial divisor. If the original number is returned then that number is prime. Key 99991 ENTER↑ 2 and XEQ "NP". 99,991 is returned after about 41 seconds and hence 99,991 is prime.

MORE EXAMPLES OF NP

Example 4: Find all the prime factors of 4,019,788,151.

The starting trial divisor will be 2.

Do:	See:	Result:
4019788151 ENTER↑ 2		enter initial inputs
XEQ "NP"	37	first prime factor
R/S	89	second prime factor
R/S	163	third prime factor
R/S	7,489	fourth prime factor
R/S	1	routine finished

4,019,788,151 = 37*89*163*7489.

APPLICATION PROGRAM 1 FOR NP

The following routine called PNG for Prime Number Generator makes use of NP to generate prime numbers. Input to this routine is an odd number which serves as the starting point for the search for primes. If a printer is connected the generated primes will be printed.

```
LBL*PNG
  2
LBL 01
XROM NP
  X=Y?
 VIEW X
  CLX
   2
 ST + Y
GTO 01
```

Key in any odd number and XEQ "PNG". The following list of primes was obtained by keying in 3 and XEQ "PNG". Press R/S to end the routine when you are tired of looking at prime numbers.

LIST OF PRIMES

3	101	229	373	521	673	839
5	103	233	379	523	677	853
7	107	239	383	541	683	857
11	109	241	389	547	691	859
13	113	251	397	557	701	863
17	127	257	401	563	709	877
19	131	263	409	569	719	881
23	137	269	419	571	727	883
29	139	271	421	577	733	887
31	149	277	431	587	739	907
37	151	281	433	593	743	911
41	157	283	439	599	751	919
43	163	293	443	601	757	929
47	167	307	449	607	761	937
53	173	311	457	613	769	941
59	179	313	461	617	773	947
61	181	317	463	619	787	953
67	191	331	467	631	797	967
71	193	337	479	641	809	971
73	197	347	487	643	811	977
79	199	349	491	647	821	983
83	211	353	499	653	823	991
89	223	359	503	659	827	997
97	227	367	509	661	829	1009

LIST OF LARGE PRIMES

9,999,999,967	9,999,999,673
9,999,999,943	9,999,999,661
9,999,999,929	9,999,999,631
9,999,999,881	9,999,999,619
9,999,999,851	9,999,999,557
9,999,999,833	9,999,999,511
9,999,999,817	9,999,999,491
9,999,999,787	9,999,999,479
9,999,999,781	9,999,999,379
9,999,999,769	9,999,999,371
9,999,999,727	9,999,999,367
9,999,999,707	9,999,999,337
9,999,999,703	9,999,999,319
9,999,999,701	9,999,999,253
9,999,999,679	9,999,999,241

List of large primes found by Richard Nelson (1) using calls to NP.

APPLICATION PROGRAM 2 FOR NP

Use NP to help evaluate the Euler Phi-function φ(n), the number of integers smaller than and relatively prime to n. Two integers are called relatively prime if there is no prime number which is a common factor of both integers. An equivalent mathematical description is that the greatest common divisor of the two integers is 1.

The φ function is useful in the arithmetic of residues modulo an integer n, or in the structures of cyclic groups of n elements. For example, if an integer m is relatively prime to n, it is invertible modulo n and is a generator of the cyclic group of n elements. φ(n) is also the number of invertible residues mod n, or the number of generators of a cyclic group of n elements.

A closed form for φ is given by:

$\phi(0) = 0$ by convention
$\phi(1) = 1$ by convention
$\phi(p^k) = p^{k-1}(p-1)$ if p is prime
$\phi(m*n) = \phi(m)*\phi(n)$ if m & n relatively prime

The program "PHN" given here will determine φ(n) where n is the absolute value of the integral part of the number found in the X-register. In addition to the stack, it uses two extra registers M and N (these alpha registers may be replaced by two ordinary registers if desired). Register M contains the accumulation of a product which eventually builds up to φ(n). Register N carries successive prime factors of n. If a prime factor repeats, it is immediately multiplied to the product in the M register and the factorization continues via NP ; if the factor is new, the factor decreased by 1 is multiplied to the quantity in the M register. This is accomplished by a DSE X, which also detects the end of the factorization of n.

BAR CODE ON PAGE 480

```
APPLICATION PROGRAM FOR: NP

01♦LBL "PHN"        15♦LBL 03
02♦LBL C            16 ST* [
03 INT              17 R↑
04 ABS              18 R↑
05 X=0?             19 XROM "NP"
06 RTN              20 ST/ Y
07  E               21 ENTER↑
08 X=Y?             22 X<> \
09 RTN              23 RCL Y
10 STO [            24 X≠Y?
11 ENTER↑           25 DSE X
12 STO \            26 GTO 03
13 ENTER↑           27 RCL [
14 ST+ Z            28 RTN
```

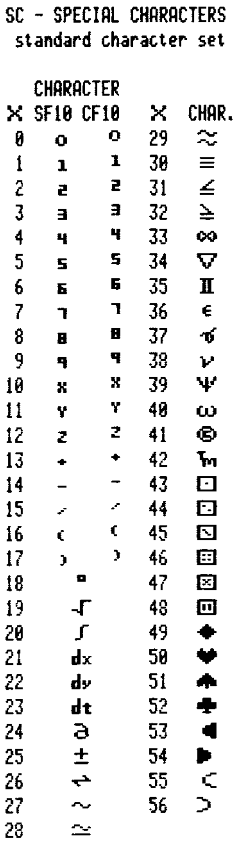
Examples:
$\phi(2) = 1$
$\phi(17) = 16$
$\phi(41) = 40$
$\phi(697) = \phi(17*41) = 16*40 = 640$
$\phi(289) = \phi(17*17) = 17*16 = 272$

```
Routine Listing For: NP

 98♦LBL e           112 X<> L
 99♦LBL "NP"        113 2
100 RCL Y           114 X=Y?
101 SQRT            115 SIGN
102 LASTX           116 +
103 X<> Z           117 GTO 09
104♦LBL 09          118♦LBL 10
105 X>Y?            119 R↑
106 R↑              120 LASTX
107 R↑              121 X>Y?
108 X<>Y            122 ENTER↑
109 MOD             123 RTN
110 X=0?            124 ST/ Y
111 GTO 10          125 GTO e
```

APPENDIX L - SPECIAL CHARACTERS - SC

This routine was planned to be included in the ROM until it was replaced by the matrix routines M1 through M5, which were felt to be more useful. Special Characters extends the 82143A printer standard character set by an additional 60 characters. These include subscripts, superscripts, math and game symbols, and more frequently used greek letters. A complete list of all the symbols included appears in table 1, below.



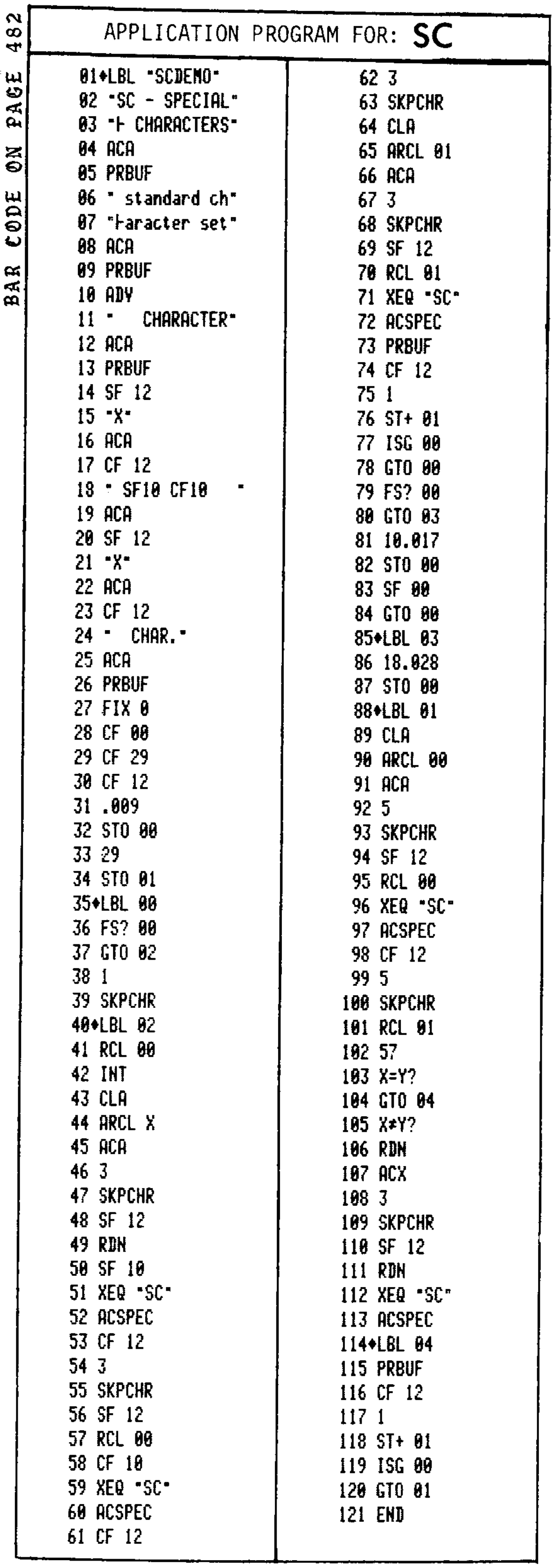
SC - SPECIAL CHARACTERS
standard character set

X	CHARACTER SF10	CHARACTER CF10	X	CHAR.
0	o	o	29	≈
1	1	1	30	≡
2	2	2	31	≤
3	3	3	32	≥
4	4	4	33	∞
5	5	5	34	∇
6	6	6	35	Π
7	7	7	36	∈
8	8	8	37	σ
9	9	9	38	ν
10	x	x	39	Ψ
11	y	y	40	ω
12	z	z	41	©
13	+	+	42	™
14	-	-	43	⚀
15	/	/	44	⚁
16	(	(	45	⚂
17	)	)	46	⚃
18	▫		47	⚄
19	-∫		48	⚅
20	∫		49	♦
21	dx		50	♥
22	dy		51	♠
23	dt		52	♣
24	∂		53	◀
25	±		54	▶
26	⇄		55	⊂
27	~		56	⊃
28	≃			

Table 1. The complete list of new symbols added to the printer's standard ACCHR set by use of the Special Characters routine. This table was produced by the 'SCDEMO' program.

Note that the first 18 characters, numbered 0 through 17, produce superscripts if F10 is clear and subscripts if F10 is set. The remainder, characters 18 through 56, are unaffected by the status of flag 10. A listing of the program to produce table 1 is presented here:

BAR CODE ON PAGE 482



APPLICATION PROGRAM FOR: SC

```
01♦LBL "SCDEMO"
02 "SC - SPECIAL"
03 "⊦ CHARACTERS"
04 ACA
05 PRBUF
06 " standard ch"
07 "⊦aracter set"
08 ACA
09 PRBUF
10 ADV
11 "   CHARACTER"
12 ACA
13 PRBUF
14 SF 12
15 "X"
16 ACA
17 CF 12
18 " SF10 CF10   "
19 ACA
20 SF 12
21 "X"
22 ACA
23 CF 12
24 "  CHAR."
25 ACA
26 PRBUF
27 FIX 0
28 CF 00
29 CF 29
30 CF 12
31 .009
32 STO 00
33 29
34 STO 01
35♦LBL 00
36 FS? 00
37 GTO 02
38 1
39 SKPCHR
40♦LBL 02
41 RCL 00
42 INT
43 CLA
44 ARCL X
45 ACA
46 3
47 SKPCHR
48 SF 12
49 RDN
50 SF 10
51 XEQ "SC"
52 ACSPEC
53 CF 12
54 3
55 SKPCHR
56 SF 12
57 RCL 00
58 CF 10
59 XEQ "SC"
60 ACSPEC
61 CF 12
62 3
63 SKPCHR
64 CLA
65 ARCL 01
66 ACA
67 3
68 SKPCHR
69 SF 12
70 RCL 01
71 XEQ "SC"
72 ACSPEC
73 PRBUF
74 CF 12
75 1
76 ST+ 01
77 ISG 00
78 GTO 00
79 FS? 00
80 GTO 03
81 10.017
82 STO 00
83 SF 00
84 GTO 00
85♦LBL 03
86 18.028
87 STO 00
88♦LBL 01
89 CLA
90 ARCL 00
91 ACA
92 5
93 SKPCHR
94 SF 12
95 RCL 00
96 XEQ "SC"
97 ACSPEC
98 CF 12
99 5
100 SKPCHR
101 RCL 01
102 57
103 X=Y?
104 GTO 04
105 X≠Y?
106 RDN
107 ACX
108 3
109 SKPCHR
110 SF 12
111 RDN
112 XEQ "SC"
113 ACSPEC
114♦LBL 04
115 PRBUF
116 CF 12
117 1
118 ST+ 01
119 ISG 00
120 GTO 01
121 END
```

The barcode for both SC and SCDEMO appear in appendix K of this manual.

APPENDIX L CONTINUED ON PAGE 429.

APPENDIX L CONTINUED FROM PAGE 423.

COMPLETE INSTRUCTIONS FOR SC

Since this is a RAM program, one must first load it into the 41C, either by scanning the barcode or reading magnetic cards recorded earlier. Now, each time a character from the set in SC is desired, one needs only to place the character number into X and XEQ SC. The synthetic text string corresponding to the special printer character will be placed in X, to either be placed immediately into the print buffer by ACSPEC, or stored in a data register for later use.

The first 18 characters may be printed as either superscripts (by clearing flag 10) or subscripts (by setting flag 10). If flag 10 is set while accessing a character which has only one form (characters #18 - #56), the character will not be printed correctly. Therefore F10 should remain clear during the use of these characters. After F10 is set before executing SC, the flag is automatically cleared so no characters are accidentally modified.

One efficient use of SC would be to load all the desired characters into data registers first, and then to recall them when needed. An example of this would be if a program using the symbols of the six faces of dice. Once the text strings are in six registers, they are later recalled and ACSPEC'ed into the print buffer. Thus the SC program would only have to be called once for each different character desired, rather than each time the character was required.

A convenient routine for exploring the special characters is labeled PSC below:

```
01  LBL PSC        04  STOP
02  XEQ  SC        05  PRBUF
03  ACSPEC         06  RTN
```

Read the SC program into the HP-41. Key the routine and assign LBL PSC to a key. ENTER the number of the symbol and press the PSC key. If you want it to print, press R/S. Build up the buffer with up to six SC symbols (no spaces) using the 82143A printer.

Example 1. Print the following lines on the printer using the SC program:

H2O ⟷ H+ + OH-
∫eX dx = eX
-b + SQR (b2-4ac)/2a
(Print the 4 phases of the moon)

```
01◆LBL "H2O"        14 "H"
 02 SF 12           15 ACA
 03 "H"             16 CF 10
 04 ACA             17 13
 05 SF 10           18 XEQ "SC"
 06 2               19 ACSPEC
 07 XEQ "SC"        20 "+OH"
 08 ACSPEC          21 ACA
 09 "O"             22 14
 10 ACA             23 XEQ "SC"
 11 26              24 ACSPEC
 12 XEQ "SC"        25 PRBUF
 13 ACSPEC          26 END
```

$H_2O \rightleftharpoons H^+ + OH^-$

```
01◆LBL "eX"         13 SKPCOL
02 SF 12            14 21
03 20               15 XEQ "SC"
04 XEQ "SC"         16 ACSPEC
05 ACSPEC           17 " = e"
06 "e"              18 ACA
07 ACA              19 10
08 CF 10            20 XEQ "SC"
09 10               21 ACSPEC
10 XEQ "SC"         22 PRBUF
11 ACSPEC           23 END
12 2
```

$\int e^x dx = e^x$

```
01◆LBL "QU"         12 ACSPEC
02 CF 12            13 "(b"
03 "-b "            14 ACA
04 ACA              15 CF 10
05 25               16 2
06 XEQ "SC"         17 XEQ "SC"
07 ACSPEC           18 ACSPEC
08 " "              19 "-4ac)/2a"
09 ACA              20 ACA
10 19               21 PRBUF
11 XEQ "SC"         22 END
```

$-b \pm \sqrt{}(b^2\text{-}4ac)/2a$

```
01◆LBL "PH"         14 ACSPEC
02 CF 12            15 1
03 "MOON PHASES:"   16 SKPCOL
04 PRA              17 RDN
05 53               18 ACSPEC
06 XEQ "SC"         19 55
07 ACSPEC           20 XEQ "SC"
08 1                21 ACSPEC
09 SKPCOL           22 56
10 RDN              23 XEQ "SC"
11 ACSPEC           24 ACSPEC
12 54               25 PRBUF
13 XEQ "SC"         26 END
```

```
MOON PHASES:
◖●◗○
```

FURTHER DISCUSSION OF SC

For those who do not wish to load the entire 500-plus bytes of SC into RAM memory each time a handful of special characters is desired, the barcodes below will suffice. These are the data barcodes for the individual characters, which can be scanned directly into the ALPHA register or into a program line. WARNING: Many of these codes will not operate correctly if scanned in normal mode. Those containing bytes from row zero of the hex table will usually lock up the calculator when scanned if not in program mode. They do operate correctly, however, as program lines.

The codes, below, which lock up the 41C if scanned in normal mode are marked with an asterisk.

APPENDIX L CONTINUED ON PAGE 439.

APPENDIX L CONTINUED FROM PAGE 429.

In addition, certain of the data barcodes are 7-byte text lines. These load into ALPHA in such a way that a RCL M instruction is required to bring it into X for correct accumulation into the print buffer. The other, shorter text lines may be placed into X by ASTO X, since the lines do not contain information in the first byte, which includes the nybble which is the sign of the mantissa. In the barcodes to follow, those marked ´M´ require RCL M and those unmarked require ASTO X before ACSPEC.

LINE BY LINE ANALYSIS OF SC

Lines 01 through 06 determine which text line is to be placed in the X register, by a computed branch to a numeric label.

Lines 07 through 14 determine whether the text string is to be brought into the X register via RCL M (if the text line is 7 characters long) or by ASTO X (if the string is shorter than 7 characters). In addition, for the characters which may be superscripts or subscripts, 2 null bytes are appended if flag 10 is set, converting the superscript string to a subscript.

Lines 15 through 197 consist of the 57 individual subroutines which place the text lines into ALPHA.

BAR CODE ON PAGE 482

Routine Listing For: SC

```
01♦LBL "SC"
02 15
03 XROM "QR"
04 20
05 ST+ Z
06 XEQ IND Z
07 FS?C 10
08 "├♦♦"
09 RCL [
10 SF 25
11 CHS
12 FS?C 25
13 ASTO X
14 RTN
15♦LBL 20
16 XEQ IND Y
17 RTN
18♦LBL 00
19 "ₓH"
20 RTN
21♦LBL 01
22 "α0"
23 RTN
24♦LBL 02
25 "↓J"
26 RTN
27♦LBL 03
28 "βJ"
29 RTN
30♦LBL 04
31 "ₓ⌶"
32 RTN
33♦LBL 05
34 "β"
35 RTN
36♦LBL 06
37 "↓"
38 RTN
39♦LBL 07
40 "@"
41 RTN
42♦LBL 08
43 "↓"
44 RTN
45♦LBL 09
46 "ₓ"
47 RTN
48♦LBL 10
49 "Γ£"
50 RTN
51♦LBL 11
52 "←"
53 RTN
54♦LBL 12
55 "ΓJ"
56 RTN
57♦LBL 13
58 "ₓ↓α"
59 RTN
60♦LBL 14
61 "ₓx̄α"
62 RTN
63♦LBL 21
64 XEQ IND Y
65 RTN
66♦LBL 00
67 "x̄x̄x̄"
68 RTN
69♦LBL 01
70 "←"
71 RTN
72♦LBL 02
73 "Δ"
74 RTN
75♦LBL 03
76 "p ♦"
77 RTN
78♦LBL 04
79 "@⌶
@"
80 RTN
81♦LBL 05
82 "x̄α↓@"
83 RTN
84♦LBL 06
85 "ΩGδōH"
86 RTN
87♦LBL 07
88 "ΩGāθō"
89 RTN
90♦LBL 08
91 "ΩGx̄>H"
92 RTN
93♦LBL 09
94 "ₓ2♦"
95 RTN
96♦LBL 10
97 "x̄$K"♦"
98 RTN
99♦LBL 11
100 " x̄Δ"
101 RTN
102♦LBL 12
103 "  ĀαΔΔ"
104 RTN
105♦LBL 13
106 "δ$ȧ$"
107 RTN
108♦LBL 14
109 "ȧQ"""
110 RTN
111♦LBL 22
112 XEQ IND Y
113 RTN
114♦LBL 00
115 "ₓR     J♦"
116 RTN
117♦LBL 01
118 "x̄
119 RTN
120♦LBL 02
121 "x̄*(♦"
122 RTN
123♦LBL 03
124 "◂ΩAβΩe"
125 RTN
126♦LBL 04
127 "μkō,F"
128 RTN
129♦LBL 05
130 "x̄⌶?
"
131 RTN
132♦LBL 06
133 "ₓJ♦"
134 RTN
135♦LBL 07
136 "@G≠ₓ"
137 RTN
138♦LBL 08
139 "'αΓ♦"
140 RTN
141♦LBL 09
142 "μa⌶Γ←"
143 RTN
144♦LBL 10
145 "$↓0"8"
146 RTN
147♦LBL 11
148 "◂Ā
Ωe"
149 RTN
150♦LBL 12
151 "α├μαx"
152 RTN
153♦LBL 13
154 "Ωμ00ᵀ"
155 RTN
156♦LBL 14
157 "ΩμX4ᵀ"
158 RTN
159♦LBL 23
160 XEQ IND Y
161 RTN
162♦LBL 00
163 "ΩμY4ᵀ"
164 RTN
165♦LBL 01
166 "Ω∡X5ᵀ"
167 RTN
168♦LBL 02
169 "Ω∡Y5ᵀ"
170 RTN
171♦LBL 03
172 "Ω∡7ᵀ"
173 RTN
174♦LBL 04
175 " Δ"
176 RTN
177♦LBL 05
178 "8"
179 RTN
180♦LBL 06
181 "8"
182 RTN
183♦LBL 07
184 "pe"
185 RTN
186♦LBL 08
187 "ₓ "
188 RTN
189♦LBL 09
190 "Ω♦♦"
191 RTN
192♦LBL 10
193 "ₓH"
194 RTN
195♦LBL 11
196 "ΩΓ♦$G♦♦"
197 END
```

REFERENCES FOR SC

See PPC Calculator Journal, V7N10P11b.

APPENDIX L CONTINUED ON PAGE 443.

APPENDIX L CONTINUED FROM PAGE 443.

CONTRIBUTORS HISTORY FOR SC

The SC program was originally writted by Jake Schwartz (1820), and was modified by Roger Hill (4940) to reduce the byte count significantly. Additional assistance for choice and design of the special printer characters was provided by John McGechie (3324), Earnest Gibbs (4610), William Wimsatt (5807) and Randall Pratt (2860).

FINAL REMARKS FOR SC

This program exemplifies the value of the wand as a device for creation of 41C synthetic program lines. In conjunction with the printer, the wand and barcode can provide new character sets for almost any special application. The characters chosen for the SC program were those which were felt to be most useful to the people who would have the PPC ROM. When the PPC Barcode Book is produced, we will be able to further exploit the advantages of scanning synthetic text lines dirtectly into HP41C program memory.

Synthetic text lines in the SC program:

```
08: Append 2 nulls      46: F3 01 C2 9F
19: F4 01 C4 48 8E      49: F3 06 C2 1B
22: F3 04 4F 90         52: F2 CE 03
25: F3 07 4A 97         55: F3 06 4A 93
28: F3 05 4A 9F         58: F3 01 07 04
34: F3 01 C2 1F         61: F3 01 02 04
34: F3 05 CA 9D         67: F3 02 02 02
37: F3 07 CA 9D         70: F3 03 88 80
40: F2 40 9F            73: F2 08 8E
43: F3 07 CA 9F         76: F4 70 A1 C0 00
79: F6 02 04 07 C0 40 80
82: F6 40 82 0F E0 40 81
85: F7 11 E2 47 F0 12 18 48
88: F7 11 E2 47 F0 16 10 18
91: F7 11 E2 47 F0 02 3E 48
94: F6 01 8C 99 32 9E 00
97: F6 02 24 4B F1 22 00
100: F6 20 C2 81 02 86 08
103: F6 20 20 41 04 08 08
106: F6 91 12 24 8A 14 24
109: F6 88 89 14 51 22 22
115: F6 01 52 A5 4A 95 00
118: F6 02 85 8A 94 A8 80
121: F6 02 8D 2A 96 28 00
124: F6 71 11 41 05 11 1C
127: F6 0C 6B 18 2C 46 83
130: F6 02 0F F8 3F E0 80
133: F5 01 C5 4A 80 00
136: F6 20 23 C8 8E 80 80
139: F5 27 84 04 06 00
142: F6 0C 61 0F E4 06 03
145: F6 E2 24 07 10 22 38
148: F6 71 15 DA B5 51 1C
151: F6 04 7F 91 0C 04 78
154: F7 11 FE 0C 19 30 60 FF
157: F7 11 FE 0C 58 34 60 FF
163: F7 11 FE 0C 59 34 60 FF
166: F7 11 FE 0D 58 35 60 FF
169: F7 11 FE 0D 59 35 60 FF
172: F7 11 FE 0D D8 37 60 FF
175: F6 20 E3 EF EF 8E 08
178: F6 38 FB EF 8F 8F 8E
181: F6 30 F3 CF EF 0F 0C
184: F6 70 E5 FF F7 CE 1C
187: F5 01 C7 CF BF FF
190: F7 11 FF FB E7 C7 00 00
193: F5 01 C4 48 A0 C1
196: F7 11 06 0A 24 47 00 00
```

TECHNICAL DETAILS

RAM ROUTINE	SC	SIZE: 000

Stack Usage: 0 T: USED 1 Z: USED 2 Y: USED 3 X: USED 4 L: USED	Flag Usage: 04: NOT USED 05: NOT USED 06: NOT USED 07: NOT USED 08: NOT USED 09: NOT USED 10: USED 25: USED
Alpha Register Usage: 5 M: USED 6 N: NOT USED 7 O: NOT USED 8 P: NOT USED	
Other Status Registers: 9 Q: 10 ⊢: NONE USED 11 a: 12 b: 13 c: 14 d: 15 e:	Display Mode: ANY Angular Mode: ANY Unused Subroutine Levels: 3
ΣREG: NOT USED Data Registers: R00: R06: NONE USED R07: R08: R09: R10: R11: R12:	Global Labels Called: Direct: QR Secondary: NONE Local Labels In This Routine: 00 to 14, 20 to 23

Execution Time:

Less than 3 seconds.

Peripherals Required:

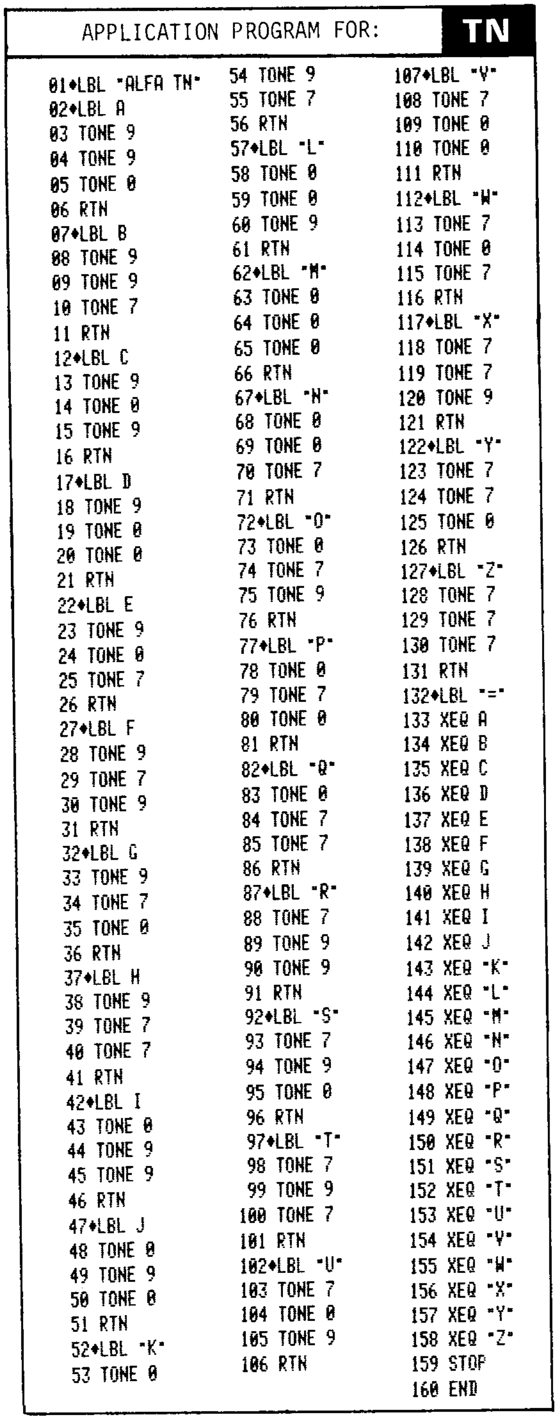
None to run SC, but printer required to print char's

Interruptible? YES Execute Anytime? YES Program File: N/A Bytes In RAM: 518 Registers To Copy: N/A	Other Comments: Use to load data registers with special characters, then RCL and ACSPEC later when they are needed.

APPLICATION PROGRAM 3 FOR TN

TOM is experimenting with a voice recognition program on his HP-85. The program and interfacing hardware is really an amplitude/time waveform recognition system that is "taught" specific sound patterns. After studying the synthetic tones on the HP-41, Tom wonders if he could have the HP-41 "talk" to the HP-85. Looking over the HP-41 TONE table, Tom selected a three tone system using the short duration tones, TONE 70, TONE 87, and TONE 89. Three tones in combinations of three provide 3^3=27 different codes. This is adequate for the 26 letters of the alphabet. Using this concept, Tom, wrote the ALFA TN program shown below. Each letter routine is assigned to its corresponding key for demonstration and test purposes. A full alphabet sequence is accomplished by calling all 26 routines one after another. This is done under Label "=" at line 132.

BAR CODE ON PAGE 485



APPLICATION PROGRAM FOR: TN

```
01♦LBL "ALFA TN"
02♦LBL A
03 TONE 9
04 TONE 9
05 TONE 0
06 RTN
07♦LBL B
08 TONE 9
09 TONE 9
10 TONE 7
11 RTN
12♦LBL C
13 TONE 9
14 TONE 0
15 TONE 9
16 RTN
17♦LBL D
18 TONE 9
19 TONE 0
20 TONE 0
21 RTN
22♦LBL E
23 TONE 9
24 TONE 0
25 TONE 7
26 RTN
27♦LBL F
28 TONE 9
29 TONE 7
30 TONE 9
31 RTN
32♦LBL G
33 TONE 9
34 TONE 7
35 TONE 0
36 RTN
37♦LBL H
38 TONE 9
39 TONE 7
40 TONE 7
41 RTN
42♦LBL I
43 TONE 0
44 TONE 9
45 TONE 9
46 RTN
47♦LBL J
48 TONE 0
49 TONE 9
50 TONE 0
51 RTN
52♦LBL "K"
53 TONE 0
54 TONE 9
55 TONE 7
56 RTN
57♦LBL "L"
58 TONE 0
59 TONE 0
60 TONE 9
61 RTN
62♦LBL "M"
63 TONE 0
64 TONE 0
65 TONE 0
66 RTN
67♦LBL "N"
68 TONE 0
69 TONE 0
70 TONE 7
71 RTN
72♦LBL "O"
73 TONE 0
74 TONE 7
75 TONE 9
76 RTN
77♦LBL "P"
78 TONE 0
79 TONE 7
80 TONE 0
81 RTN
82♦LBL "Q"
83 TONE 0
84 TONE 7
85 TONE 7
86 RTN
87♦LBL "R"
88 TONE 7
89 TONE 9
90 TONE 9
91 RTN
92♦LBL "S"
93 TONE 7
94 TONE 9
95 TONE 0
96 RTN
97♦LBL "T"
98 TONE 7
99 TONE 9
100 TONE 7
101 RTN
102♦LBL "U"
103 TONE 7
104 TONE 0
105 TONE 9
106 RTN
107♦LBL "V"
108 TONE 7
109 TONE 0
110 TONE 0
111 RTN
112♦LBL "W"
113 TONE 7
114 TONE 0
115 TONE 7
116 RTN
117♦LBL "X"
118 TONE 7
119 TONE 7
120 TONE 9
121 RTN
122♦LBL "Y"
123 TONE 7
124 TONE 7
125 TONE 0
126 RTN
127♦LBL "Z"
128 TONE 7
129 TONE 7
130 TONE 7
131 RTN
132♦LBL "="
133 XEQ A
134 XEQ B
135 XEQ C
136 XEQ D
137 XEQ E
138 XEQ F
139 XEQ G
140 XEQ H
141 XEQ I
142 XEQ J
143 XEQ "K"
144 XEQ "L"
145 XEQ "M"
146 XEQ "N"
147 XEQ "O"
148 XEQ "P"
149 XEQ "Q"
150 XEQ "R"
151 XEQ "S"
152 XEQ "T"
153 XEQ "U"
154 XEQ "V"
155 XEQ "W"
156 XEQ "X"
157 XEQ "Y"
158 XEQ "Z"
159 STOP
160 END
```

Tom used a voice input TIC-TAC-TOE game on the HP-85 to test the concept. He used a bender coupler-amplifier speaker on the HP-41 and executed the sequence of ten codes (A-J) as digit inputs to the HP-85. Much to everyones amazement, it actually worked. Perhaps those tones have some use after all.

APPLICATION PROGRAM 4 FOR TN

This program is used with a bender coupler and tone detector that "outpulses" a relay on the telephone line for dialing purposes. The operating philosophy of the program is to prompt for a NAME? of six characters. Once a name is input, R/S causes the program to "look up" the seven digit telephone number and produce a short tone sequence for each digit. Five produces five short tones, Nine produces nine tones, Zero ten tones, etc. The "fall through" label scheme used allows a fast "pulse". This is too fast for most local offices, but is easily slowed down.

Label "DIAL" is assigned to the "D" key. ENTER is assigned to the "C" key. To dial press "D". Key NAME? after prompt, then R/S. If a new number is to be added, press "C", followed by PRGM. The Line "25 LBL:NAME?" serves as a prompt to key in a new number in the format shown below.

```
LBL ABCDEF      (up to 6 characters)
.NNNNNNN        (7 digits, could be up to
GTO 11                   10, see line 11)
```

A-F is the Alpha name, and .NN is the seven digit telephone number entered as a decimal. The GTO 11 instruction actually does the "dialing". Two telephone numbers are in the program for demonstration purposes. They may be deleted. The ? entry may be used to time a particular HP-41 for dialing speed.

Here is a line by line description of the program.

The label at line 01 provides a display description of what the key does. Line 02 is a local label used to save bytes, because it is addressed twice--lines 22 and 76. The CLX at line 03 insures that the SIN at line 06 operates on zero. Lines 04 and 05 display "AUTO DIALER" briefly while the SIN of zero is calculated. Lines 03 and 06 simply provide a delay. The NAME? prompt is displayed in lines 07 thru 09. The NAME is assumed in ALPHA when R/S causes program resumption at line 10 where ALPHA is turned off. The ISG value 0.006 is stored in R00 and the entered name is stored in the X register in lines 11 thru 13. The display shows SEARCHING using lines 14 and 15. ROM routine VA is used at lines 05, 15, 19, and 34 as good practice, even though the printer is not to be connected when using this program. Flag 25 is set at line 16 in case the indirect GTO at line 17 can't be executed. If a nonexistent label is searched for, line 17 is "skipped" and a "CAN'T FIND" display is shown by lines 18 and 19. A two second low frequency tone (TONE 30) at line 20 provides a notice of failure to find the name and a fixed duration of the CAN'T FIND display. The GTO 13 at line 21 restarts the program.

If the global label is found at line 17, the routine format is to enter the telephone number into X and go to LBL 11 at line 31. The clear flag 25 instruction at line 32 is included as good practice to avoid too wide a window of a non-indicating error

```
145 GTO 00
146♦LBL 07
147 X<> d
148 "USED"
149♦LBL 00
150 AVIEW
151 TONE 2
152 GTO 22
153♦LBL "NN"
154 CLA
155 PROMPT
156♦LBL 08
157 INT
158 256
159 MOD
160 LASTX
161 +
162 OCT
163 X<> d
164 FS?C 11
165 SF 12
166 FS?C 10
167 SF 11
168 FS?C 09
169 SF 10
170 FS?C 07
171 SF 09
172 FS?C 06
173 SF 08
174 X<> d
175 ASTO [
176 X<> [
177 "├*"
178 STO \
179 "├*"
180 X<> \
181 CLA
182 STO [
183 RTN
184 GTO 08

185♦LBL 09
186 CF 09
187 CF 10
188 CF 19
189 CF 20
190 CF 21
191 SF 29
192 AVIEW
193 RCL c
194 STO [
195 "├♦♦♦♦x̄"
196 RCL [
197 X<> d
198 CF 00
199 CF 01
200 CF 02
201 CF 03
202 FS?C 07
203 SF 05
204 FS?C 08
205 SF 06
206 FS?C 09
207 SF 07
208 FS?C 10
209 SF 09
210 FS?C 11
211 SF 10
212 FS?C 12
213 SF 11
214 X<> d
215 DEC
216 2
```

```
217 -
218 191
219 X<>Y
220 X<Y?
221 GTO 05
222  E3
223 /
224 +
225 "*i♦8♦"
226 RTN

227♦LBL "CA"
228 "CLEAR A"
229 XEQ 09
230 STO 00
231 RCL [
232 X<> c
233 .
234 STO 09
235 STO 14
236 RCL b
237 X<>Y
238 STO IND T
239 X<>Y
240 ISG T
241 STO b
242 RDN
243 RDN
244 X<> c
245 GTO 25

246♦LBL 04
247♦LBL "PA"
248 "PACK A"
249 XEQ 09
250 ENTER↑
251 FIX 8
252  E
253 -
254 .
255 RCL [
256 GTO 12

257♦LBL 10
258 CF 20
259 X<> c
260 X<>Y
261 X<> IND Z
262 X<>Y
263 X<> c
264 X<>Y
265 CLA
266 ARCL [
267 STO [
268 SIGN
269 X≠0?
270 GTO 01
271 "├****"
272 ASHF
273 X<> \
274 X=0?
275 SF 19
276 X<> L
277 STO [
278 ASHF
279 X<> [
280 X=0?
281 SF 20
282 FC? 19
283 FS? 20
284 GTO 02
285 X<> [
286 ""
```

```
287 ARCL X
288 X<> [

289♦LBL 11
290 X<>Y
291 X<> c
292 X<>Y
293 X<> IND T
294 X<>Y
295 X<> c
296 ISG T

297♦LBL 12
298 ISG Z
299 GTO 10
300 X<>Y

301♦LBL 01
302 FC?C 10
303 GTO 03
304 SF 09
305 CLA

306♦LBL 13
307 X<> [
308 ""
309 X<> [
310 X<> \
311 ASTO [
312 ASTO [
313 ARCL 00
314 "├****"
315 X<> \
316 GTO 11
```

```
317♦LBL 02
318 X<>Y
319 FS?C 19
320 FC? 20
321 FS? 54
322 GTO 12
323 "├*"
324 FC?C 20
325 "├***"
326 X<>Y
327 CLX
328 STO \
329 FS?C 10
330 GTO 13
331 SF 10
332 ASTO 00
333 X<>Y
334 GTO 12

335♦LBL 03
336 R↑
337 ENTER↑
338 INT
339 192
340 -
341 .5
342 FC? 09
343 SIGN
344 +
345 TONE 0
346 CLA
347 FIX 1
348 ARCL X
349 "├ REG USED"
350 AVIEW
351 .END.
```

MKA Melbourne

This key assignment program appeared in PPC CJ, V7N10P19. It was written by Tom Cadwallader (3502) and modified slightly by John McGechie (3324) and Richard Collett (4523). When using MKA you need not make an even number of key assignments, but you must press R/S in response to the prompt for the second of each pair of key assignments. No data registers are used, so SIZE 000 is OK. The same user constraints apply as for KA#18 (END above MKA in Catalog 1, don't SST, don't mix assignments with MK).

Note on MKA listing: The following lines are not represented accurately in the printed listing. Their hexadecimal equivalents are given here.

Line	Hex Equivalent
08	F7 F0 00 00 00 00 00 00
144	F1 F0
285	F5 F0 04 01 C9 01
393	F1 F0
404	F1 F0
416	F1 F0

FURTHER ASSISTANCE ON MK

Call Tom Cadwallader (3502) at (406) 727-6869.
Call Roger Hill (4940) at (618) 656-8825.

BAR CODE ON PAGE 486

```
01•LBL "MKA"
02•LBL 17
03 SF 03

04•LBL 04
05 CF 00
06 CF 01
07 XEQ 00
08 "••••••"
09 X<> [
10 X<> IND L

11•LBL 03
12 ""
13 X<> [
14 "├•"
15 X<> \
16 X=0?
17 GTO 01
18 "├------"
19 X<> \
20 ISG Z
21 GTO 02
22 STO IND L
23 RDN
24 STO c
25 FC?C 03
26 GTO 18
27 XEQ "PKA"
28 GTO 04

29•LBL 02
30 X<> IND Z
31 GTO 03

32•LBL 01
33 RDN
34 STO c
35 CLA
36 SF 02
37 CF 03

38•LBL 15
39 CF 22
40 ASTO L

41•LBL 19
42 "PRE↑POST↑KEY"
43 TONE 9
44 PROMPT
45 CLA
46 ARCL L
47 FC? 22
48 GTO 16
49 X<> Z
50 4
51 RDN
52 X=0?
53 X<> T
54 XEQ "D↑C"
55 XEQ "D↑C"
56 36
57 STO a
58 RDN
59 ENTER↑
60 X<0?
61 SF 00
62 ABS
63 .1
64 *
65 LASTX
66 -
67 INT
68 8
69 X<Y?
70 GTO 01
71 RDN
72 ST- a
73 4
74 X=Y?
75 SF 04
76 RDN
77 LASTX
78 FRC
79 X=0?
80 CF 04
81 .1
82 FC?C 04
83 CLX
84 +
85 80
86 *
87 ST- a
88 2
89 *
90 +
91 8
92 FC? 00
93 CLX
94 +
95 X<> a
96 X<0?
97 GTO 01
98 24
99 X<=Y?
100 SF 01
101 X>Y?
102 CLX
103 -
104 FS? 00
105 RCL e
106 FC? 00
107 RCL ↑
108 ASTO L
109 STO [
110 FS? 01
111 "├•••"
112 X<> [
113 X<> d
114 FC? IND Y
115 GTO 05
116 X<> d
117 CLA
118 ARCL L
119 "├TAKEN"
120 TONE 0
121 GTO 02
            |
122•LBL 05
123 SF IND Y
124 X<> d
125 STO [
126 "├••••"
127 FC?C 01
128 "├•••"
129 RCL \
130 FS? 00
131 STO e
132 FC?C 00
133 STO ↑
134 CLA
135 ARCL L
136 RCL a
137 XEQ "D↑C"
138 FS?C 02
139 GTO 15

140•LBL 16
141 FS? 02
142 GTO 19
143 ASTO X
144 ""
145 FC?C 22
146 "├•••"
147 ARCL X
148 RCL [
149 XEQ 11
150 176
151 R↑
152 STO IND Y
153 R↑
154 X<> c
155 CLST
156 "R/S TO CONT-"
157 BEEP
158 PROMPT
159 GTO 17

160•LBL 01
161 "├ERROR"

162•LBL 02
163 AVIEW
164 TONE 0
165 X<> \
166 CLA
167 X<> [
168 ASTO L
169 GTO 15

170•LBL 18
171 CLST
172 TONE 0
173 " NO ROOM"
174 PROMPT
175 GTO 18

176•LBL "NNN"
177 CLA

178•LBL 07
179 PROMPT
180 XEQ "D↑C"
181 RCL [
182 GTO 07

183•LBL "D↑C"
184 INT
185 OCT
186 X=0?
187 GTO 06
188 STO ]
189 RDN
190 4 E2
191 ST+ ]
192 X<> ]
193 X<> d
194 FS?C 11
195 SF 12
196 FS?C 10
197 SF 11
198 FS?C 09
199 SF 10
200 FS? 07
201 SF 09
202 FS? 06
203 SF 08
204 SF 03
205 ARCL d
206 STO d
207 "├**"
208 CLX
209 STO ↑
210 X<> ]
211 X<> \
212 STO [
213 RDN
214 RTN

215•LBL 06
216 "├•"
217 STO ↑
218 STO ]
219 RDN
220 RTN

221•LBL "CKA"
222•LBL 22
223 XEQ 00

224•LBL 08
225 CLX
226 X<> IND Z
227 SF 25
228 X=0?
229 FS?C 25
230 GTO 09
231 ISG Z
232 GTO 08

233•LBL 09
234 RDN
235 STO c
236 CLX
237 STO ↑
238 STO e
239 BEEP
240 RTN

241•LBL "SAX"
242 XEQ 10
243 RDN
244 X<>Y
245 X<> IND L
246 GTO 09

247•LBL "RAX"
248 XEQ 10
249 RDN
250 RCL IND L
251 FC? 25
252 ENTER↑

253•LBL 09
254 X<>Y
255 X<> c
256 RDN
257 FS?C 25
258 RTN
259 SF 99

260•LBL 10
261 16
262 -
263 ABS
264 RDN
265 GTO 11

266•LBL "C16"
267 XEQ 11
268 RDN
269 RTN

270•LBL 00
271 XEQ "FEA"
272 193
273 -
274 X<0?
275 GTO 18
276 176
277 +
278  E3
279 /
280 176
281 +
282 ENTER↑

283•LBL 11
284 XEQ 12
285 "α*i*"
286 X<> [
287 STO \
288 "├AB"
289 X<> \
290 ENTER↑
291 X<> c
292 X<>Y
293 SF 25
294 RTN

295•LBL "GTE"
296 XEQ 12
297 X<> d
298 SF 02
299 SF 03
300 X<> d
301 CLA
302 STO [
303 "├AB"
304 X<> \
305 STO b

306•LBL "FEA"
307 XEQ 12
308 X<> d
309 FS?C 07
310 SF 05
311 FS?C 08
312 SF 06
313 FS?C 09
314 SF 07
315 FS?C 10
316 SF 09
317 FS?C 11
318 SF 10
319 FS?C 12
320 SF 11
321 X<> d
322 DEC
323 RTN

324•LBL 12
325 CLA
326 RCL c
327 X<> [
328 "├••••x̄"
329 X<> [
330 X<> d
331 CF 00
332 CF 01
333 CF 02
334 CF 03
335 X<> d
336 RTN

337•LBL "PKA"
338 XEQ 00
339 CLA
340 CF 03

341•LBL 20
342 FS?C 03
343 GTO 21
344 CLX
345 X<> IND Z
346 SF 25
347 X=0?
348 FS?C 25
349 GTO 21
350 ASTO a
351 CF 01
352 STO [
353 ASHF
354 X<> [
355 X=0?
356 SF 01
357 CLX
358 "├•"
359 STO \
360 "├••"
361 X<> \
362 "├•"
363 X<> \
364 X=0?
365 GTO 13
366 X<> \
367 ASTO [
368 "├•"
369 STO \
370 "├•"
371 X<> \

372•LBL 13
373 STO \
374 FC?C 01
375 "├•••"
376 X<> \
377 CLA
378 STO [
379 ASTO X
380 CLA
381 ARCL a
382 ARCL X
383 SF 03
384 ISG Z
385 CF 03
386 CLX
387 RCL \
388 X=0?
389 GTO 20
390 ASTO X
391 ASHF
392 ASTO a
393 ""
394 ARCL X
395 CLX
396 X<> [
397 STO IND T
398 ARCL a
399 ISG T
400 GTO 20
401 RTN

402•LBL 21
403 X<> [
404 ""
405 X<> [
406 "├↑"
407 X<> \
408 "├↑↑↑↑↑↑"
409 X<> \
410 X<> IND L
411 SIGN
412 X≠0?
413 GTO 14
414 CLX
415 LASTX
416 ""
417 X<> [
418 "├↑"
419 X<> \
420 "├↑↑↑↑↑↑"
421 X<> \
422 STO IND T

423•LBL 14
424 RDN
425 STO c
426 RDN
427 INT
428 176
429 -
430 X=0?
431 GTO 22
432 RDN
433 INT
434 175
435 -
436 BEEP
437 .END.
```

BAR CODE ON PAGE 483

ALTERNATE VERSION OF: VK

```
 01♦LBL "VK"
 02 8
 03 STO a
 04 SF 25
 05 PRBUF
 06 FC? 25
 07 CF 21
08 "KEYS USED:"
 09 AVIEW
10 "0α♦♦♦♦♦"
 11 RCL '
 12 X<> [
 13 RCL e
 14 X<> \
 15 STO ]
 16 FIX 3
 17 CF 29
 18 ARCL Y
 19 FIX 0
 20 X<> ]
 21 STO [
  22 "├ "
 23 X<> \
 24 X<> d
 25 RCL [
 26 CLA

 27♦LBL 01
 28 X<> Z
 29 ABS
 30 FRC
 31 CHS
 32 LASTX
 33 INT
 34 +
 35 39
 36 -
 37 X<> Z

 38♦LBL 02
39 FC? IND Z
 40 FC? 50
 41 GTO 15
 42 X<> d
43 FC? IND Z
 44 FC? 50
 45 GTO 15
 46 X<> d

 47♦LBL 03
 48 ISG Z
 49 GTO 02
 50 DSE a
 51 GTO 01
 52 X<>Y
 53 STO d
 54 FS? 21
 55 CLA
 56 "├ END"
 57 AVIEW
 58 GTO 14

 59♦LBL 15
 60 FC? 50
 61 GTO 15
```

```
 62 X<> d
 63 STO \
 64 X<> Z
 65 STO [
 66 INT
 67 123
 68 +
 69 8
 70 /
 71 INT
 72 LASTX
 73 FRC
 74 80
 75 *
 76 +
 77 41
 78 X<Y?
 79 DSE Y
 80 3
 81 +
 82 X<Y?
 83 ISG Y
 84 ""
 85 FS? 42
86 FC? IND [
 87 CHS
 88 X<>Y
 89 ABS
 90 X<> [
 91 R↑
 92 RCL \
 93 " "
 94 FC? 42
 95 "    -"
 96 X<>Y
 97 X<> d
 98 ARCL L
 99 ISG T
100 GTO 13
101 "├ -"
102 ARCL L

103♦LBL 13
104 AVIEW
105 FC? 21
106 TONE 0
107 X<> d
108 X<>Y
109 FS? 42
110 X<> d
111 GTO 03

112♦LBL 15
113 RDN
114 STO d
115 CLD

116♦LBL 14
117 FC?C 25
118 SF 21
119 CLST
120 FIX 2
121 END
```

Routine Listing For: VK

```
 01♦LBL "VK"
 02 SF 21
 03 FS? 55
 04 PRKEYS
 05 FS? 55
 06 RTN
 07 CF 21
 08 8
 09 RCL '
 10 XEQ 07
  11 "├0"
 12 X<> [
 13 X<> d
 14 RCL e
 15 XEQ 07
  16 "├ "
 17 X<> Z
 18 X<> [

 19♦LBL 01
20 -27.00008
 21 RCL [
 22 -
 23 STO \
 24 RDN

 25♦LBL 02
26 FC? IND \
 27 FC? 50
 28 GTO 05
 29 X<> d
30 FC? IND \
 31 FC? 50
 32 GTO 05
 33 X<> d

 34♦LBL 03
 35 ISG \
 36 GTO 02
 37 DSE [
 38 GTO 01
 39 X<>Y

 40♦LBL 04
 41 STO d
 42 CLST
 43 CLA
 44 PSE
 45 CLD
 46 RTN

 47♦LBL 05
 48 X<> d
 49 35
 50 RCL \
 51 INT
 52 +
 53 OCT
 54 1
 55 ST+ Y
 56 %
 57 +
 58 10
```

```
 59 MOD
 60 LASTX
 61 *
 62 INT
 63 STO a
 64 43
 65 -
 66 ABS
 67 1
 68 X<Y?
 69 ST+ a
 70 FS? 42
71 FC? IND \
 72 CHS
 73 ABS
 74 X<> [
 75 RDN
 76 X<> \
 77 RDN
 78 " "
 79 FC? 42
 80 "    -"
 81 FS? 50
 82 X<> d
 83 X<>Y
 84 FC? 50
 85 GTO 04
 86 X<> d
 87 X<> _
 88 CLX
 89 RCL d
 90 FIX 0
 91 CF 29
 92 ARCL a
 93 ISG L
 94 GTO 06
 95 "├ -"
 96 ARCL a

 97♦LBL 06
 98 STO d
 99 X<> _
100 AVIEW
101 TONE 0
102 R↑
103 STO \
104 R↑
105 STO [
106 RDN
107 RDN
108 X<> d
109 X<>Y
110 FS? 42
111 X<> d
112 GTO 03

113♦LBL 07
114 CLA
115 X<> [
116 "├*****"
117 X<> \
118 X<> [
119 RTN
```

APPENDIX A - ADVANCED APPLICATIONS OF LR/SR & HD/UD

HANOI TOWER PUZZLE GENERALIZED

Given: $\underline{m}$ pegs, $\underline{n}$ discs of varying size stacked in order of size (large on the bottom, small on the top) on peg 1.

Problem: In the smallest number of moves, one disc at a time, in such a way that a disc is never placed on top of a smaller one, move the $\underline{n}$ discs (similarly stacked) from peg 1 to peg m.

A brief glance at the original 3-peg problem (Tower of Hanoi) will prove helpful. The crux of the solution to this simplest version is the need to uncover the bottom disc, which in turn leads to the need to transfer $\underline{n-1}$ discs to peg 2. This perspective leads to a repeated reduction by one of the number of discs to be moved, until we are led to the need to move only one disc. The immediately preceding problem was the need to move 2 discs; and the solution has become: first move the top disc to the other peg, then move the bottom disc to the target peg, and finally move the top disc again. The top disc was moved twice. If there were 3 discs to move, the top disc would be moved twice in loading the alternate peg, and then twice more in unloading to the target peg. The inductive argument shows that if there are $\underline{n}$ discs to move, the top disc undergoes 2^{n-1} moves, the disc below undergoes 2^{n-2} moves, etc., while the bottom disc requires only $2^{n-n} = 1$ move. There are easier ways of establishing the total number of moves ($1+2$ $2+2^2+\ldots+2^{n-1}=2^n-1$), but this way of looking at it has value for resolving the problem with more than 3 pegs.

We need to explicitly note some parallel features of the m-peg version of this puzzle.

(A) If using $\underline{m}$ pegs, we have $\underline{m-2}$ pegs (pegs 2,3,..., m-1) to temporarily hold the top $\underline{n-1}$ discs while we move the bottom disc to peg m.

(B) Unpacking the top $\underline{n-1}$ discs can be viewed as $\underline{m-2}$ subtasks to be performed sequentially:

(1) Using all $\underline{m}$ pegs, first load n_m discs onto peg 2.

(2) Using $\underline{m-1}$ pegs (by the rules peg 2 can't be used), load n_{m-1} discs onto peg 3.

(3) Using $\underline{m-2}$ pegs (pegs 2 and 3 can't be used), load n_{m-2} discs onto peg 4.

⋮

(m-2) Using 3 pegs (pegs m, 1, and m-1), load n_3 discs onto peg m-1.

(C) After moving the bottom disc to peg m, unload the substacks in a sequence opposite to the loading:

(1) Using 3 pegs (m-1, 1, and m) transfer the n_3 discs on peg m-1 to peg m.

(2) Using 4 pegs (m-2, 1, m-1, and m) transfer the n_4 discs on peg m-2 to peg m.

⋮

(m-2) Using all $\underline{m}$ pegs transfer the n_m discs on peg 2 to peg m.

(D) Note that unloading a peg to the target peg entails the same number of moves as loading the peg from the original stack.

Again we can show that the number of moves any disc undergoes in arriving at its final location is a power of 2, but the reasoning is more complicated than when we assume that m=3.

Suppose we want to know the number of moves required to place the top disc of a stack of $\underline{n}^{(0)}$ discs when we're using $\underline{m}$ pegs. By observation B, we know that if $\underline{n}^{(0)} > 1$, our first subtask is to move $\underline{n}^{(1)}$ discs to peg 2, where $\underline{n}^{(1)} < \underline{n}^{(0)}$. If $\underline{n}^{(1)} > 1$, we can proceed to the first subtask of the first subtask, and that would entail the movement of $\underline{n}^{(2)}$ discs to some peg, where $\underline{n}^{(2)} < \underline{n}^{(1)}$. By this recursion, we arrive at the transfer of the top disc to some peg. By observation D, unraveling this recursion leads to repeated doubling of the number of moves undergone by the top disc. Of course, when $\underline{m} > 3$, the total number of moves of the top disc of a stack containing $\underline{n}$ discs is less than 2^{n-1}. The disc below will require either as many or half as many moves. Let's proceed to make this more definite.

Let $Z_n(m)$ = number of moves required to transfer n discs using m pegs. Clearly $Z_1(m) = 1$. How does $Z_{n+1}(m) - Z_n(m)$ behave?? Certainly $Z_2(m) - Z_1(m) = 2$, since the top disc has to be placed on and removed from an intermediate peg. Obviously, in fact, $Z_{n+1}(m) - Z_n(m)$ remains 2 up to n = m-2, there being $\underline{m-2}$ intermediate pegs available. At this point we're out of intermediate pegs, so at least one disc will require more than one intermediate resting place. By observation D and the preceding discussion, that disc will require 4 moves. As $\underline{n}$ increases, $Z_{n+1}(m) - Z_n(m)$ eventually becomes 8, still later 16, etc. To be more specific, we need an appropriate recursive relationships.

Let Q(m,e) = the maximum $\underline{n}$ such that $Z_n(m) - Z_{n-1}(m) = 2^e$. In order to extend this definition to Q(m,0), let $Z_0(m) = 0$. Suppose we know Q(μ,e) for μ = 3 to $\underline{m}$, and let $\underline{N} = 1 + \Sigma\, Q(\mu,e)$, (where the summation is from μ=3 to μ=m), be the number of discs we will transfer using $\underline{m}$ pegs. Using the strategy noted in observations $\underline{B}$ and C:

(1) Transfer Q(m,e) discs to peg 2 from peg 1.

(2) Transfer Q(m-1,e) discs to peg 3 from peg 1.

APPENDIX A CONTINUED ON PAGE 37

APPENDIX A CONTINUED FROM PAGE 33.

$\vdots$

(m-2) Transfer Q(3,e) discs to peg m-1 from peg 1.
(m-1) Transfer 1 disc to peg m from peg 1.
( ) Transfer Q(3,e) discs to peg m from peg m-1.
(m+1) Transfer Q(4,e) discs to peg m from peg m-2.

$\vdots$

(2m-3) Transfer Q(m,e) discs to peg m from peg 2.

We see that no disc required more than $2 \cdot 2^e = 2^{e+1}$ moves. On the other hand, transferring N+1 discs would have required that one disc undergo $2 \cdot 2^{e+1} = 2^{e+2}$ moves. In other words, N = Q(m,e+1). We've established that

$$Q(m,e+1) = 1 + \Sigma\{Q(\mu,e):\mu=3,\ldots,m\}$$

But then

$$Q(m,e) = [1 + \Sigma\{Q(\mu,e-1): =3,\ldots,m-1\}] + Q(m,e-1)$$

$$= Q(m-1,e) + Q(m,e-1)$$

A table of values for Q(m,e) will reveal the simple pattern:

e \ m	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8
2	3	6	10	15	21	28	36
3	4	10	20	35	56	84	120
4	5	15	35	70	126	210	330
5	6	21	56	126	252	462	792

We now have all we need to know to solve our problem. Suppose, for example, we wish to move 25 discs using 6 pegs. This calls for partitioning the upper 24 discs into four substacks in a way which will minimize the total number of required moves. If we look at the Q(m,e) table, we see that the second row of values for m = 3, 4, 5, 6 contains 2, 3, 4, 5 totaling 14, while the third row contains 3, 6, 10, 15 totaling 34. Thus, we see that an optimum strategy requires as many as $2^3 = 8$ moves, but never more, for some discs. Any combination of four numbers n_3, n_4, n_5, n_6 such that $Q(i,1) \le n_i \le Q(i,2)$ and $\Sigma\{n_i:i=3,4,5,6\} = 24$ will suffice for a partitioning corresponding to a minimum number of moves. Note that in general there is more than one solution to a given problem. In our sample problem

$$n_3 = 2,\ n_4 = 3,\ n_5 = 10,\ n_6 = 9$$

will work as well as

$$n_3 = 3,\ n_4 = 6,\ n_5 = 10,\ n_6 = 5,$$

to mention but two of 56 possibilities.

Each of these subproblems (e.g., move $n_5 = 10$ discs using 5 pegs) can be handled similarly, until the requirement is reduced to moving a single disc.

To evaluate the number of moves required for a specific problem is straightforward: simply add up the number of moves required for each disc. Consider, for instance, our example of moving 25 discs using 6 pegs:

24-(2+3+4+5)=10 discs each	requiring	$2 \cdot 2^2$ moves	-- 80
14-(1+1+1+1)=10 discs each	requiring	$2 \cdot 2^1$ moves	-- 40
4 discs each	requiring	$2 \cdot 2^0$ moves	-- 8
1 disc	requiring	1 move	-- 1

for a total of 129 moves.

As a second example, consider moving 13 discs using 5 pegs:

12-(2+3+4)=3 discs each	requiring	$2 \cdot 2^2$ moves	-- 24
9-(1+1+1)=6 discs each	requiring	$2 \cdot 2^1$ moves	-- 24
3 discs each	requiring	$2 \cdot 2^0$ moves	-- 6
1 disc	requiring	1 move	-- 1

for a total of 55 moves.

The recursive routine GHT (Generalized Hanoi Tower) implements the strategy outlined for solving the m-peg version of this puzzle. Such a routine would probably be regarded as outside the scope of the HP-41 were it not for the curtain-moving and return-stack extension routines provided by the PPC Custom ROM. Naturally the memory limitations of the HP-41 impose some constraints, but cases requiring more memory than is available are also, for the most part, cases entailing too many moves for recreational interest. The data compaction schemes employed in GHT do not permit m>9 nor n>45. The number r of data registers required for legal values of m and n is given by

$$r = 9n_3 + mp + 2e + \max(6, m + 1)$$

where:

n_3 = number of discs (as evaluated by 'PARTS') to be moved to peg m-1 using only 3 pegs;

p = number of data registers allocated to each peg = $\lceil(n/5)$;

e = number of required extensions of the return stack = $\lfloor[(n_3-1)/5]$;

$\lceil z$ = least integer not less than z ('ceiling' of z);
$\lfloor z$ = greatest integer not greater than z ('floor' of z).

As long as SIZE ≥ 4, set-up routine IGT will proceed successfully, issuing prompting messages if more data registers are needed. The calling sequence is

# of pegs ↑ # of discs, XEQ 'IGT'.

If resizing prompts are displayed, resize as requested and press R/S to continue. When "READY?" is displayed, all required data for calling GHT have been established. At this point you have the option of turning on the printer to record the successive moves; press R/S to continue.

Each call on recursive routine GHT (except the first) is preceded by a call on HD to hide 9 data registers:

00 for curtain moving: set up by HD; used by UD
01 $.i_1i_2{-}{-}{-}i_{m'}$ = indices of pegs currently in use
02 n' = # of discs currently being moved
03 m' = # of pegs currently in use
04 $W.pm_0$ = global work area specification
05 subtask control: peg count
06 subtask control: peg indices

APPENDIX A CONTINUED ON PAGE 61.

APPENDIX A CONTINUED FROM PAGE 37.

07 partition data
08 also partition data, if # of parts > 5

The global work area is accessible to GHT regardless of the depth of recursive call ($\le n_3-1$). The specification $W.pm_0$ is a compact storage of three items of information needed by MOVE (the subroutine for moving one disc to peg Y from peg X) and SHOW (the subroutine for displaying the current distribution of discs on pegs):

W = pointer to global work area

p = # of data registers allocated to each peg

m_0 = original number of pegs (m passed to IGT)

Partition data is in a compact form ($a_1a_2b_1b_2\ldots$), a pair of decimal digits to each part, beginning with number of discs to be moved using 3 pegs, and ending with the number of pegs to be moved using m'-2 pegs. (This data is set up in lines 03 through 22 of GHT; register 08 is only needed when m'-2 > 5 or m' > 7, but to avoid the logic overhead GHT always uses 9 registers per recursive call.) The position of the decimal point varies during the process. When loading the intermediate pegs, the decimal point moves to the left; it moves back to the right when the intermediate pegs are being unloaded.

The global work area is allocated as follows:

W: move counter, initialized to -1

W+1 → W+p: discs for peg 1

W+p+1 → W+2p: discs for peg 2

⋮

$W+(m_0-1)p+1 \rightarrow W+m_0p$: discs for peg m_0

Discs are designated by integers from 1 to n, where i<j whenever disc i is smaller than disc j. Each disc designation i is kept in compact storage (at most 5 to a register) as two decimal digits $d_{i1}\ d_{i2}$.

IGT initializes the work area and R01 through R04, given the number (m) of pegs in register Y and the number (n) of discs in register X:

.12---m ---> R01

n ---> R02

m ---> R03

W.pm ---> R04

-1 ---> R_W

$d_{11}d_{12}d_{21}d_{22}{---}_{51}d_{52}$ ---> R_{W+1}

⋮

$---d_{n1}d_{n2}$ --- R_{W+p}

(in fact, IGT begins with a CLRG, so any register not explicitly addressed in IGT starts out with a zero value.) Additionally IGT calls on IXR to initialize return stack management. (See Application Program 1 in LR description for further details regarding IXR.) Note that the pointer in R13 is set to 9 less than pointer in R04 before GHT calls itself. (See lines 231 through 234.) Of course, before a call on itself GHT must also set up R10 through R12 which become R01 through R03 after the curtain is raised. (Lines 51 through 83 do this during the loading of the intermediate pegs; lines 139 through 193 do the same task for the unloading process.)

Finally, to avoid loss of a return path as a consequence of excessive subroutine nesting, GHT calls LRR upon entry and SRR just before exit. No other calls for safeguarding the return path are necessary, since GHT does not initiate any other chain of calls more than two deep. (See Application Program 1 in LR description for further details regarding LRR and SRR.) However, a brief examination of the Q(m,e) table will show that the two cases with the smallest number of moves that require an extension of the return stack ($n_3 \ge 6$) are m = 3 and n = 7 or 8, which entail 127 and 255 moves respectively. Other stack-extending cases are far more prolonged. If you plan to avoid such time consuming cases (by restricting yourself to cases where $n_3 < 6$) you can avoid the execution overhead of IXR, LRR, and SRR by removing lines 80 through 81 in IGT, lines 02 and 218 through 219 in GHT, and replace 'GTO 15' in line 165 of GHT by "RTN'.

The logic of 'MOVE' and 'SHOW' (disc stacks are displayed from top to bottom), although using some tedious housekeeping to unravel compact storage, can be gleaned by careful perusal of the listing, keeping in mind the allocation scheme already described. However, a few words about 'PARTS' are needed to ease comprehension of its logic.

If we examine the Q(m,e) table, a simple method for evaluating the optimum distribution of discs on intermediate pegs quickly becomes apparent. We'll use an earlier example of m=6 and n=25 to keep our description concrete. 'PARTS' builds up the partitioning using R09 through R(6+m), which would be R09 through R12 in our example. We begin with all parts set to zero, and the count k of discs to distribute to n-1=24.

```
REPEAT WHILE k > 0:
    INC ← 1
    REPEAT FOR j = 9 through 12:
        $R_j$ ← $R_j$ + INC
        k ← k - INC
        INC ← $R_j$
        IF k=0, EXIT
        IF k<0, $R_j$ ← $R_j$ + k and EXIT
```

The following table shows the changing states of R09 through R12 and of k:

R09	R10	R11	R12	k
0	0	0	0	24
1	0	0	0	23
1	1	0	0	22
1	1	1	0	21
1	1	1	1	20
2	1	1	1	19
2	3	1	1	17
2	3	4	1	14
2	3	4	5	10
3	3	4	5	9
3	6	4	5	6
3	6	10	5	0

A final word of caution. You may want to abort program execution for some reason. If you note your size (via S? , e.g.) before execution, then XEQ S? if you stop the program before it finishes execution, subtract the original size, and call CU to re-establish communication with all your data registers.

APPENDIX A CONTINUED ON PAGE 133.

APPENDIX A CONTINUED
FROM PAGE 61.

```
01♦LBL "IGT"
02 CLRG
03 STO 02
04 X<>Y
05 STO 03
06 8
07 +
08 XROM "VS"
09 FC?C 25
10 PROMPT
11 RCL 03
12 STO 04
13 0

14♦LBL 00
15 RCL Y
16 +
17  E1
18 /
19 DSE Y
20 GTO 00
21 STO 01
22 RCL 02
23 5
24 /
25 ENTER↑
26 INT
27 X=Y?
28 GTO 01
29 1
30 +

31♦LBL 01
32 STO 00
33 XEQ "PARTS"
34 RCL 00
35 STO 05
36 RCL 09
37  E1
38 ST/ 04
39 ST/ 04
40 ST/ 05
41 DSE X
42 *
43 RCL 03
44 2
45 -
46 ST+ Y
47 3
48 -
49 X<=0?
50 ST- Y
51 RDN
52 STO 06
53 RCL 05
54 +
55 ST+ 04
56 INT
57 RCL 03
58 RCL 00
59 *
60 +
61 1
62 +
63 STO 07
64 RCL 09
65 1
66 -
67 5
68 /
69 INT
70 2
71 ST* Y
72 +
73 +
74 XROM "VS"
75 FC?C 25
76 PROMPT
77 RCL 06
78 -1
79 STO IND Y
80 RCL 07
81 XEQ "IRX"
82 RCL 06
83 RCL 02
84  E3
85 /
86 1
87 ST+ Z
88 +
89 .5
90 ST+ Z

91♦LBL 05
92 CLX
93 5
94 STO 05
95 CLX

96♦LBL 06
97  E2
98 *
99 RCL Y
100 INT
101 +
102 DSE 05
103 GTO 07
104 STO IND Z
105 ISG Z
106 ISG Y
107 GTO 05
108 GTO 10

109♦LBL 07
110 ISG Y
111 GTO 06
112 STO IND Z

113♦LBL 10
114 "READY?"
115 PROMPT
116 XEQ "SHOW"
117 GTO "GHT"
118 END
```

```
01♦LBL "GHT"
02 XEQ "LRR"
03 XEQ "PARTS"
04 7.5
05 ENTER↑
06 5
07 X<> 06
08 0

09♦LBL 02
10  E2
11 *
12 RCL IND Y
13 +
14 DSE 06
15 GTO 03
16 STO IND Z
17 X<> 06
18 ISG Z

19♦LBL 03
20 ISG Y
21 GTO 02
22 STO IND Z
23 RCL 03
24 2
25 -
26 STO 05
27 RCL 01
28  E1
29 *
30 INT
31 STO 06
32 RCL 03
33 STO [
34 10↑X
35 RCL 01
36 *
37 DSE [

38♦LBL 04
39  E1
40 ST* 06
41 /
42 ENTER↑
43 FRC
44  E1
45 *
46 INT
47 ST+ 06
48 RDN
49 DSE [
50 GTO 04

51♦LBL 05
52 5
53 RCL 05
54 X>Y?
55 GTO 06
56 RCL 07
57  E2
58 /
59 STO 07
60 GTO 07

61♦LBL 06
62 RCL 08
63  E2
64 /
65 STO 08

66♦LBL 07
67 FRC
68  E2
69 *
70 INT
71 X=0?
72 GTO 09
73 XEQ 70
74 RCL 06
75 RCL 05
76 2
77 +
78 FS?C 01
79 GTO 08
80 STO 12
81 10↑X
82 /
83 STO 10
84 XEQ 75
85 XEQ "GHT"
86 XROM "UD"
87 GTO 09

88♦LBL 08
89 10↑X
90 /
91  E1
92 *
93 INT
94 RCL 06
95  E1
96 /
97 FRC
98  E1
99 *
100 X<>Y
101 XEQ "MOVE"

102♦LBL 09
103 RCL 06
104  E1
105 /
106 INT
107 STO 06
108 DSE 05
109 GTO 05
110 RCL 01
111 RCL 03
112 1
113 -
114 10↑X
115 *
116 FRC
117  E1
118 *
119 RCL 01
120  E1
121 *
122 INT
123 XEQ "MOVE"
124 RCL 03
125 2
126 -
127 STO 06
128  E3
129 /
130 1
131 ST+ 06
132 +
133 STO 05
134 RCL 06
135 10↑X
136 RCL 01
137 *
138 STO 06

139♦LBL 10
140 5
141 RCL 05
142 INT
143 X>Y?
144 GTO 11
145 RCL 07
146 RCL 07
147  E2
148 *
149 STO 07
150 GTO 12

151♦LBL 11
152 RCL 08
153 RCL 08
154  E2
155 *
156 STO 08

157♦LBL 12
158 INT
159 X<>Y
160 INT
161  E2
162 *
163 -
164 X=0?
165 GTO 15
166 XEQ 70
167 RCL 06
168  E1
169 /
170 STO 06
171 FRC
172  E1
173 *
174 FRC
175 LASTX
176 INT
177 RCL 01
178  E1
179 ST* Z
180 *
181 INT
182 +
183 +
184 FS?C 01
185 GTO 13
186  E2
187 /
188 STO 10
189 RCL 05
190 INT
191 2
192 +
193 STO 12
194 XEQ 75
195 XEQ "GHT"
196 XROM "UD"
197 GTO 14

198♦LBL 13
199 INT
200 LASTX
201 FRC
202 RCL 05
203 INT
204 1
205 -
206 10↑X
207 *
208 FRC
209  E1
210 ST/ Z
211 *
212 X<>Y
213 INT
214 XEQ "MOVE"

215♦LBL 14
216 ISG 05
217 GTO 10

218♦LBL 15
219 XEQ "SRR"
220 RTN

221♦LBL 70
222 CF 01
223 1
224 X<>Y
225 X=Y?
226 SF 01
227 FC? 01
228 STO 11
229 RTN

230♦LBL 75
231 RCL 04
232 STO 13
233 9
234 ST- 13
235 XROM "HD"
236 END
```

```
01♦LBL "PARTS"
02 RCL 02
03 STO 05
04 1
05 ST- 05
06 RCL 03
07 2
08 -
09  E3
10 /
11 +
12 8.008
13 +
14 STO 06
15 STO 07
16 0

17♦LBL 00
18 STO IND 07
19 ISG 07
20 GTO 00

21♦LBL 01
22 1
23 RCL 06
24 STO 07

25♦LBL 02
26 X<>Y
27 ST+ IND 07
28 ST- 05
29 RCL IND 07
30 RCL 05
31 X>0?
32 GTO 03
33 X=0?
34 RTN
35 ST+ IND 07
36 RTN

37♦LBL 03
38 ISG 07
39 GTO 02
40 GTO 01
41 END
```

```
01♦LBL "MOVE"
02  E2
03 STO 10
04 RDN
05 XEQ 20
06 ST* Y
07 ST* Z
08 ST- Z
09 RDN
10 RCL 04
11 INT
12 ST+ Y
13 1.5
14 +
15 ST+ Z
16 RDN
17 STO [
18 X<>Y
19 STO \
20 0
21 STO 09

22♦LBL 01
23 RCL IND [
24 X=0?
25 GTO 03
26 XEQ 25
27 10↑X
28 STO Z
29 /
30 ENTER↑
31 INT
32 X<> 09
33 CF 00
34 X=0?
35 SF 00
36 STO T
37 RDN
38 FRC
39 *
40 FS?C 00
41 GTO 02
42 RCL 10
43 *
44 +

45♦LBL 02
46 STO IND [

47♦LBL 03
48 DSE [
49 DSE ]
50 GTO 01
51 RCL 09
52 STO [
53 XEQ 20
54 CF 00

55♦LBL 05
56  E8
57 RCL IND
58 X<Y?
59 SF 00
60 RCL [
61 STO Z
62 RDN
63 FS? 00
64 GTO 06
65 RCL X
66 RCL 10
67 MOD
68 STO [
69 RDN
70 RCL 10
71 /
72 INT

73♦LBL 06
74 X=0?
75 GTO 07
76 XEQ 25
77 2
78 +
79 10↑X
80 ST* Z
81 RDN

82♦LBL 07
83 +
84 STO IND \
85 FS?C 00
86 GTO 10
87 ISG \
88 DSE ]
89 GTO 05

90♦LBL 10
91 GTO "SHOW"

92♦LBL 20
93 RCL 04
94 FRC
95  E1
96 *
97 INT
98 STO ]
99 RTN

100♦LBL 25
101 ENTER↑
102 LOG
103 2
104 /
105 INT
106 2
107 *
108 END
```

```
01♦LBL "SHOW"
02 FIX 0
03 CF 29
04 RCL 04
05 RCL IND X
06 1
07 +
08 STO IND Y
09 X=0?
10 GTO 10
11 "---MOVE "
12 ARCL X
13 "├---"
14 BEEP
15 ADV
16 ADV
17 XROM "VA"

18♦LBL 10
19 RCL 04
20 INT
21 RCL 04
22 FRC
23  E1
24 *
25 STO 09
26 INT
27 -
28 STO 10
29 RCL 09
30 FRC
31  E2
32 /
33 1
34 +
35 STO 09

36♦LBL 00
37 ADV
38 "PEG "
39 ARCL 09
40 XROM "VA"
41 TONE 7
42 TONE 7
43 RCL 10
44 RCL 10
45 RCL 04
46 FRC
47  E1
48 *
49 INT
50 ST+ 10
51 ST+ Y
52 ST+ Z
53 +
54  E3
55 /
56 +
57 1
58 +
59 STO 11
60 RCL IND X
61 X≠0?
62 GTO 01
63 TONE 4
64 TONE 4
65 "   **EMPTY**"
66 XROM "VA"
67 GTO 04

68♦LBL 01
69 RCL IND 11
70 X=0?
71 GTO 04
72 "        "
73 ENTER↑

74♦LBL 02
75 RDN
76  E2
77 /
78 ENTER↑
79 INT
80 X≠0?
81 GTO 02
82 RDN

83♦LBL 03
84 "├."
85  E2
86 *
87 ENTER↑
88 INT
89  E1
90 X>Y?
91 "├ "
92 ARCL Y
93 RDN
94 -
95 X≠0?
96 GTO 03
97 TONE 4
98 TONE 4
99 XROM "VA"
100 ISG 11
101 GTO 01

102♦LBL 04
103 ISG 09
104 GTO 00
105 END
```

```
LBL'IGT
END          187 BYTES
LBL'GHT
END          359 BYTES
LBL'PARTS
END           68 BYTES
LBL'MOVE
END          176 BYTES
LBL'SHOW
END          190 BYTES
```

END