–BLDROM –

This rom was intended as a plug in module, unfortunately, the author could not muster the $20,000 investment for such a device. I am told that later ... HP lowered the cost, oh well. It is now being released for your enjoyment. Try out 64 functions in a 4K handcoded XROM 05.

Contents

It is recommended to print out this document, especially pages 7 and 11.
An L8/U2 eprom set with printed manual and annotated disassembly is available.

– BLDROM –

This function returns a real time constant to the three rightmost digits of the R (append or lazy T) register. Without a time module present, this function always returns 07A (122 dec).
When a time module is present, convert this number to decimal and divide it into 19260 to obtain the cpu speed in microseconds per cycle. The default, 19260/122 = 158 microseconds per cycle.

ROMA

ROM to Alpha. This function appends a rom word, address specified by X, to Alpha in base 32 (0-9,A-V). X may be decimal integer or text or NNN (NNN = Non-Normalized-Number).

GPE

Goto Program End. This programmers tool is equivalent to the following actions: CAT 1 and wait until it finishes, switch to program mode, then press Shift - BST. Essential, must have tool, can't do without.

RTN?

Is a Return pending ? HP41C "do if true" conditional. Skips one line if a return is not pending.
This simple test may seem useless, however it has great utility in user code library subroutines.
If one structures subroutines such that at entry a test is made using RTN?, the subroutine can act differently depending on wheather or not it has been called from a running program or executed from the keyboard. This structure results in subroutines which are self documenting. The structure might look like:

LBL "SUB1
RTN? GTO 00
LBL 01
prompt for input data, set up for execution, put everything in correct location, then;
LBL 00
do the purpose of subroutine assuming all required data is present and at correct locations.
RTN? RTN GTO 01

FREE?

Returns the number of registers not being used between the .END. and the IO buffer area.
BUF? returns the number of registers being used in the I/O buffer area, add 191.

AXEQ?

Alpha Execute, HP41C conditional. An attempt is made to execute the text string in Alpha, up to seven characters in length. Very similar to XEQ IND nn when nn contains a text string except XEQ IND nn is limited to six characters.
If an Alpha argument is needed, follow the name with a single space then the argument. Only programmable XROM

functions and User Code programs in ram or rom may be executed. If a User Code program called does not end with a RTN or END, the program using AXEQ? will effectively be terminated. If the function or program cannot be executed, one program line is skipped. This function was written to be a small part of an HP-IL operating system. If the program can not be found, a mass storage search of IL is performed, and if found there, loaded into ram and executed. Example input: RENAME OLDNAME,NEWNAME

BIN

This function converts the X register into a text string. If X contains a floating point number, that number is converted to binary using the absolute value of the integer portion of the number. If the X register contains a text string, this function only makes a copy in the LASTX register. If the X register contains an NNN, this function simply makes the leftmost digit equal to a one, converting the NNN into a text string. The largest integer allowed is 9,999,999,999. The old value of X is returned in LASTX and the printer is notified that X has changed. Hex is displayed if executed from the keyboard.

FPN

Generates a Floating Point Number (positive integer) in X given binary in X. The largest binary number allowed is 2540BE3FF (which results 9,999,999,999). The leftmost digit in X is ignored.
The old value of X is returned in LASTX and the printer is notified that X has changed.

BLD

BuiLD user code line. This function is a single stepping HP41C User Code assembler. It allows
assembly of standard, non-standard or precompiled lines just above the .END..

Text is keyed into the Alpha register, followed by R/S. Press R/S without an input to get back to the .END. of program memory in program mode, viewing the last line that was assembled.

Typically, writing a new program just above the .END. in program memory, when the need arises for a non-standard line: execute BLD, key in the text equivalent of the line, press R/S, press R/S and continue keying in the program. There is no need to switch out of program mode to execute BLD, it is a non-programmable function. It will be handy to have BLD assigned to a key during program development. One may find that it is less work to key standard functions into program memory using BLD versus using normal keystrokes.
If you get the TRY AGAIN error: execute SIZE and free up some room, then press R/S and continue normally.

BLD features:

Any of : " or ' (colon, double quote or single quote) is recognized as escape character.
Postfix may be number or character, missing postfix is taken as zero.
Any non-recognizable up to 15 characters is taken as Alpha Replace.
Any non-recognizable longer than 15 characters is taken as comment line.

BLD examples:

Any function line	RCL X	XEQ IND L	ST+ IND M	X<>Y	X<> L
Normal LBL, GTO or XEQ	XEQ X	LBL Z	LBL INITIALIZE I/O I		
Global LBL, GTO or XEQ	LBL :ABC:	XEQ :ABC:	XEQ :ABC is allowed		

Precompiled GTO or XEQ	GTO 5,-15	XEQ 9,150	XEQ INIT I/O 110,-47
Rom functions by name or number	ALENGXY	XROM 31,63	XROM AROT 25,6
Packed END (only use if packed)	END 9	unpacked: END 15	
Alpha Replace	:ABC:	(both escape characters are required)	
Alpha Append	::ABC		
Single or multiple number entry lines	:48	:58.423 7.23E-23	
Alpha replace Text hexadecimal	:T414243	(:T or "T or 'T)	
Literal Hexadecimal load	:HF3414243	(:H or "H or 'H)	
Comment Line	:LBL ABC	(:L or "L or 'L)	
Data names	STO MYDATA 15	RCL GROSS IND VIA 28	

BLD notes:

Precompiled XEQ and GTO use -1 as the preceeding byte of the line, and 1 as the byte following the line.
Do not use a letter label for precompiled GTO/XEQ, use the numeric equivalent instead, for example if one needs a precompiled execute 58 bytes forward at label A, use XEQ 102,58. Use only a comma for double numeric in GTO/XEQ and XROM eg: 5,-15 or 31,63. XEQ NAME is allowed if the name does not have embeded numbers.

Consecutive number entry lines must be separated by ENTER^ or put on the same line. Each space or non-numeric character between numbers will be loaded as a null byte, do not use multiple spaces between multiple number entry fields. For the text, :T and :H escape sequences, the leading bytes cannot be 00 (they will be discarded).

Do not assemble a packed END unless building a precompiled program in which case one must use one to prevent decompile. BLD always sets the final .END. to repack.

Use :T7F3A for an append colon. :ABC: is completely general, everything between the two escape characters, regardless of byte value, is converted to a text string (even leading nulls).

DISPLAY

If followed by a text string in a program, the string is loaded into the display. Alpha is not modified.
If no text string follows, up to the first 12 characters of ALPHA are displayed (no scrolling).

The text string which follows DISPLAY should be 12 characters maximum. The HP41C/CV display has 316 unique segment settings, this function uses 253 of these to display each byte with a segment setting. The [and C characters are the same segment setting. If this function is executed from the keyboard it will display a string from program memory if the user PC is positioned at a text string, otherwise it will display the Alpha register. The displayed segment settings are many times different than the standard HP41C display for a byte.

Segment setting vs byte value *

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0¢	1¢	2¢	3¢	4¢	5¢	6¢	7¢	8¢	9¢	: ¢	;¢	< ¢	= ¢	> ¢	?¢
1	sp¢	!¢	"¢	# ¢	$¢	%¢	&¢	'¢	(¢	)¢	*¢	+¢	, ¢	−¢	. ¢	/¢
2	sp	!	"	#	$	%	&	'	(	)	*	+	,	−	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	_
6	@	a	b	c	d	e	F	G	H	I	J	K	µ	≠	Σ	R
7	@¥	a ¥	b¥	c¥	d ¥	e ¡	F ¡	G¡	H¡	Y¡	Z ¡	K¡	µ ¡	≠¡	Σ¡	R ¡
8	@ ¡	A¡	B ¡	C ¥	D¥	E ¥	F ¥	G¥	H ¥	I ¥	J ¥	K¥	L¥	M¥	N ¥	O¥
9	P ¥	Q¥	R¥	S¥	T ¥	U ¥	V¥	W¥	X¥	Y ¥	Z¥	[¥	\¥	]¥	↑¥	_¥
A	sp¥	!¥	"¥	# ¥	$¥	%¥	&¥	'¥	(¥	)¥	*¥	+¥	, ¡	−¥	. ¡	/¥
B	0¥	1¥	2¥	3¥	4¥	5¥	6¥	7 ¥	8¥	9¥	: ¥	;¥	< ¥	=¥	> ¥	?¥
C	@ ,	A,	B,	C ,	D,	E ,	F ,	G,	H ,	I ,	J ,	K ,	L ,	M,	N ,	O,
D	P,	Q,	R,	S ,	T ,	U,	V,	W,	X,	Y,	Z,	[,	\ ,	] ,	↑ ,	_ ,
E	sp ,	! ,	" ,	# ,	$,	% ,	& ,	',	(,	) ,	*,	+ ,	, ,	−,	. ,	/ ,
F	0 ,	1 ,	2 ,	3 ,	4 ,	5 ,	6 ,	7 ,	8 ,	9 ,	: ,	; ,	<,	=,	>,	?,

Because the punctuation is merged with the characters, character strings are shorter than normal when punctuation is needed. Technically then, a 12 byte string could print 24 characters. Unlike AVIEW, DISPLAY only sends it's output to the printer if in NORM or TRACE. If F20 is set, DISPLAY will surpress setting the message flag and surpress sending it's output to the printer

AVU41C

Alpha View as if Alpha were an HP41C program line.

For example, put the byte 05 in Alpha and execute AVU41C; LBL 04 will be displayed. AVU41C sends it's output to the printer if NORM or TRACE. If F20 is set, AVU41C will surpress setting the message flag and surpress sending it's output to the printer.

* Some of the symbols in this table are from LCD4.TTF True Type font from hp41.org, many thanks!

BLDH

Byte LoaD Hexadecimal. Scrolling ram User Code editor. This function "inserts" bytes. This function will operate while in PRGM mode only.

Go to the line in ram program where you want to insert or change bytes, execute BLDH and the line shown will be right justified in the six byte window. To change the value of the byte at the right of the display, press backarrow and key in the new hex value on the active hex keyboard. Pressing backarrow nulls the byte currently at the right of the display. BLDH will overwrite only a null byte. Insert is always to the right of the byte at the right of the display unless the byte at the right of the display is null, in which case this null byte is overwritten. Automatic register opens are performed when needed and the repack/decompile bits are set when a register open occurs.

Use ENTER^ or R/S to exit the BLDH function, or use the top row keys: ON, Shift ON, PRGM or ALPHA.
Be carefull with the backarrow, it always nulls a byte. For example; go to the end of program memory and switch to program mode (GPE is handy for this), execute BLDH and key in F3414243 then press ENTER^. SST, BST and TAN may be used to navigate through program memory. Holding down SST or TAN, well, and USER toggles the click. Although this function has some latency, keystrokes will not be missed.

ABIN

Alpha Binary INput function. Key hex directly into the Alpha register, up to 48 digits.
Use Shift - Backarrow to clear Alpha and backarrow to delete a digit.

This function may be used from the keyboard or in a running program. Use ENTER^ (or the top row keys ON, Shift ON, PRGM, ALPHA) to exit when using the keyboard execute and R/S to continue running a program after input. In a running program, if the display contains a message, it will remain in the display until the first digit is entered. The colon signifies that the data is going into Alpha. In a running program with a message in the display, backarrow will terminate input. ENTER^ always clears the message and terminates input. Shift - Backarrow behaves normally, clearing the Alpha register and the message and continuing the colon prompt input.

XBIN

X Binary INput function. Key hex directly into X, up to 14 digits.
Use Shift - Backarrow to clear X and Backarrow to delete a digit.

Use ENTER^ (or the top row keys ON, Shift ON, PRGM, ALPHA) to exit when using keyboard execution. The right period signifies that the data is going into X. If used in a running program, use R/S to continue running the program. The normal hex keyboard is available and the numeric operator keys, EEX and CHS are mapped to hex digits also. For ABIN and XBIN, the SST/BST keys are also active and behave normally. This function, as well as the two functions above, always enable the stack lift at termination, ENTER^ does not perform a stack lift. To enter a pair of binary numbers using XBIN, enter the first number with XBIN, press ENTER^, then press the zero key then execute XBIN again.

BINHEX

BINary to HEXadecimal conversion in place in Alpha. Up to 12 bytes in Alpha are converted from binary to Intel Hex Code (0-9, A-F). Does an AVIEW for keyboard execution only.

HEXBIN

HEXadecimal to BINary conversion in place in Alpha. Up to 24 hex characters are converted to binary.
Characters must be in the range 0-9, A-F or natural hex: 0-9, : < = > ?. Keyboard: does an AVIEW.

XAH

X is converted to Hexadecimal characters (0-9, A-F) and placed in cleared Alpha.
This function is often called DECOD. Leading zeros are blanked, use ALENGXY to make the string fourteen characters long if disired (eg: 48 ENTER^ 14 ALENGXY).

AXH

Alpha to X Hexadecimal. Hexadecimal characters in Alpha are converted to binary and placed in X. This function is often called COD. It does not lift the stack and does not notify the printer.
Hexadecimal in Alpha may be 0-9, A-F or natural hex: 0-9, : < = > ?.

ACL ?

Alpha CLear ? HP41C "do if true" conditional.

ALENG ?

Returns the number of characters in Alpha to the X register. Exactly the same utility as the extended functions ALENG, however this one runs much more quickly.

ALENGX

Pads or strips characters from Alpha until ALENG is as specified by X. The space character is used for padding. If the number in X is negitive, padding/stripping is done on the Right of Alpha.

ALENGXY

Pads or strips characters from Alpha until ALENG is as specified by X. The character used for padding is specified in Y. If X is negitive, padding/stripping is performed on the Right of Alpha.
Y may be integer, Alpha or NNN.

ASTOXX

Alpha store the number of characters specified by X into X. The prior value of X is returned in LASTX. If the number in X is negitive, Alpha store is from the Right of Alpha. -6 to 6. If X has a value greater than 6, a six byte window is scanned through Alpha.

ASHFX

Shifts Alpha the number of characters specified by X. If the number in X is negitive, shifts are done on the Right of Alpha.

PRESP

Removes everything in Alpha past the first space character, including the space.

PREFX

Removes everything in Alpha past the character specified by X, including the character specified by X.

X may be integer, Alpha or NNN.

POSTSP

Removes everything in Alpha before the first space character, including the space.

POSTFX

Removes everything in Alpha before the character specified by X. The character specified by X is also removed. X may be integer, Alpha or NNN.

AXL

Alpha to X from the Left. A single character is stripped from the left of Alpha, converted to decimal and placed in X. This function only returns a zero if Alpha is clear. Respects stack lift.

AXR

Alpha to X from the right. A single character is stripped from the right of Alpha, converted to decimal and placed in X. This function returns -1 if Alpha is clear. Respects stack lift. This function is prefered if the string in Alpha may contain null characters, as AXL will discard null characters.

XAL

From X to Alpha on the Left. X may be integer (up to 9,999,999,999) or Alpha text or NNN. For integer, the integer is first converted to binary then appended to the left of Alpha. Maximum 7 characters appended.

XAR

From X to Alpha on the Right. X may be integer (up to 9,999,999,999) or Alpha text or NNN. For integer, the integer is first converted to binary then appended to the right of Alpha. Maximum 7 char appended.

AXI

Scans Alpha for a positive integer number and places it in X. If Alpha does not contain a number, -1 is returned in X. Limited to 14 signifigant digits not counting leading zeros. Stack lift is respected.

XAI

Appends a text string positive integer taken from X, to the right of alpha. Display setting does not matter, only positive integer is appended without decimal point or separators. X must be a number less than 1 E24.

Nibble Process Function

PRSS

PRSS is a nibble function language construct. PRSS takes it's argument from the string (text) which follows it in program memory or from Alpha for keyboard execution. If a string does not follow PRSS in program memory, the argument is taken from Alpha. Each nibble in the string is a function:

hex digit	mnem	operation	nibbles	width dependant
0	DECX	X ← X − 1	1	yes
1	INCX	X ← X + 1	1	yes
2	RC + Y	X ← X + Y	1	yes
3	RC − Y	X ← X − Y	1	yes
4	CLX	X ← 0	1	yes
5	X <> Y	X€ Y	1	yes
6	STOY	Y ← X	1	yes
7	OR	X ← X or Y	1	yes
8	XOR	X ← X xor Y	1	yes
9	NOT	$X \leftarrow \overline{X}$	1	yes
A	AND	X ← X and Y	1	yes
B(00 − F7)	BIT nn	binary	3	no/yes
C(0 − F)	SETX n	set X definition	2	no
D(0 − F)	LDI n	load nibble	2	yes
E(0 − F)	SHFR n	1 to 16 bits	2	yes
F(1 − E)	SETW n	set width	2	no
F0	VUHX	view X in hex	2	no
FF	SETXX	define X via X	2	no
nn for BIT (hex)			decimal	
00 − 37	CBIT 0 − 55	clear bit	0 − 55	no
40 − 77	SBIT 0 − 55	set bit	64 − 119	no
81 − B7	SHFL 1 − 55	shift left	129 − 183	yes
C1 − F7	ROTL 1 − 55	rotate left	193 − 247	yes

The width may be set from 1 to 14 nibbles (on the right of the register), 14 (FE) is the default at entry. The Y register is always defined as absolute register 2, the X register can be re-defined to be any status register using the SETX function or any absolute register using the SETXX function. The default X register is absolute register 3 (C3).

Example: rotate the rightmost 16 bits of X one bit to the left and view it: SETW 4   ROTL 1   VUXH :   F4 BC1 F0

Processor fetch is byte based. When the number of nibbles is odd, add a zero to the front of the sequence. The process will always discard a leading zero nibble. If DECX is to be the first in a sequence for PRSS, the first byte of the string would be 00, a leading null byte cannot be defined in Alpha.

A process sequence may be up to 48 nibbles for Alpha and 30 nibbles for in program sequence. A few more examples for these strings: set continuous on: CEB4B,   display user PC: CCF0,   display the .END. register: CDFFF0. Encrypt 6 text bytes in X with 6 byte text key in Y: FC 8 BD1 9 8 BD7 9. Decript: FC 9 BD9 8 9 BDF 8.

The BIT nn functions use bit numbers starting with zero on the right. Accordingly, the hex digits required are somewhat difficult to write down, a table is included on page 7.5. Try out sequences by keying nibbles into Alpha using ABIN, then execute PRSS. A word to the wise, do not leave PRSS assigned to an upper key, inadvertent keystroke with unknown in Alpha ....

7.5 Hex values for BIT nn

Flag	BIT	CBIT	SBIT	SHFL	ROTL
55	0	00	40	--	--
54	1	01	41	81	C1
53	2	02	42	82	C2
52	3	03	43	83	C3
51	4	04	44	84	C4
50	5	05	45	85	C5
49	6	06	46	86	C6
48	7	07	47	87	C7
47	8	08	48	88	C8
46	9	09	49	89	C9
45	10	0A	4A	8A	CA
44	11	0B	4B	8B	CB
43	12	0C	4C	8C	CC
42	13	0D	4D	8D	CD
41	14	0E	4E	8E	CE
40	15	0F	4F	8F	CF
39	16	10	50	90	D0
38	17	11	51	91	D1
37	18	12	52	92	D2
36	19	13	53	93	D3
35	20	14	54	94	D4
34	21	15	55	95	D5
33	22	16	56	96	D6
32	23	17	57	97	D7
31	24	18	58	98	D8
30	25	19	59	99	D9
29	26	1A	5A	9A	DA
28	27	1B	5B	9B	DB
27	28	1C	5C	9C	DC
26	29	1D	5D	9D	DD
25	30	1E	5E	9E	DE
24	31	1F	5F	9F	DF
23	32	20	60	A0	E0
22	33	21	61	A1	E1
21	34	22	62	A2	E2
20	35	23	63	A3	E3
19	36	24	64	A4	E4
18	37	25	65	A5	E5
17	38	26	66	A6	E6
16	39	27	67	A7	E7
15	40	28	68	A8	E8
14	41	29	69	A9	E9
13	42	2A	6A	AA	EA
12	43	2B	6B	AB	EB
11	44	2C	6C	AC	EC
10	45	2D	6D	AD	ED
9	46	2E	6E	AE	EE
8	47	2F	6F	AF	EF
7	48	30	70	B0	F0
6	49	31	71	B1	F1
5	50	32	72	B2	F2
4	51	33	73	B3	F3
3	52	34	74	B4	F4
2	53	35	75	B5	F5
1	54	36	76	B6	F6
0	55	37	77	B7	F7

8 Complex Functions

BLDROM provides a small set of complex arithmetic functions. Nothing fancy, only the real stack is used for a two level RPN complex stack. As one would expect, the complex X register is defined as X+jY and the complex Y register is defined as Z+jT. The complex functions modify only the complex X register. A complex X<>Y is not provided because a pair of roll ups accomplishes the same thing.

$\boxed{1/Z}$

Invert the complex X register.

$\boxed{Z+}$

Add complex X and complex Y and return result in complex X.

$\boxed{Z-}$

Subtract complex X from complex Y and return result in complex X.

$\boxed{Z*}$

Multiply.

$\boxed{Z/}$

Divide. This function is not monadic to "Out of Range" error, use 1/Z Z* to avoid this problem.
This function is not optimized for execution time (6*2436 can be done in 3*3436).

$\boxed{E\uparrow Z}$

Complex exponentiation.

$\boxed{LNZ}$

Complex natural logarithm. Principal value.

$\boxed{ENTERZ}$

Copies complex X to complex Y only.

Note that ENTERZ is not a replacement for the real ENTER^ function, it is of use to square a complex number as in ENTERZ Z*. The complex exponential function is quite general, it encompases all the complex trig and hyperbolic functions using simple complex relationships, for example :

$$\sin z = (-j/2)(e^{jz} - e^{-jz}) \qquad \sinh z = (1/2)(e^{z} - e^{-z}) \qquad Y^{X} = e^{X\,LnY}$$
$$\cos z = (1/2)(e^{jz} + e^{-jz}) \qquad \cosh z = (1/2)(e^{z} + e^{-z}) \qquad h_0^{(1)}(z) = e^{jz}/(jz)$$

A two level stack is insufficient for serious complex programming, however the real STO and RCL functions may be used as well as the two LIFO functions :

$\boxed{\uparrow Z}$

Push Z onto the LIFO stack.

POPZ

Pop Z from the LIFO stack. CLX POPZ does not modify complex Y.

Routines for adding two impedances in parallel, raising complex Y to the complex X power and complex sine :

```
LBL` PARZ    1/Z    R↑    R↑    1/Z    Z+    1/Z    RTN
LBL` ZY↑X    R↑    R↑    LNZ    Z*    E↑Z    RTN
LBL` SINZ    X<>Y    CHS    E↑Z    ENTERZ    1/Z    Z−    CHS    X<>Y    2    ST÷Z    ÷    RTN
LBLGSINHZ  R↑  R↑  ↑Z  R↑  R↑  E↑Z  ENTERZ  1/Z  Z−  2  ST÷Z  ÷  POPZ  R↑  R↑  RTN
```

9 Tone Generator

TONEZ

Tone specified by X and Y registers. The Y register specifies the tone duration (number of cycles),
the X register specifies the tone number. Tone zero is the highest frequency.

A repeat count may also be specified (multiply repeat count by 65536 and add to the number of cycles) as well as a pause count between tones (multiply the pause count by 4096 and add to the tone number).

Y = C + 65536*R
X = T + 4096*P

R	Repeat count (0 - 4095) number of tones.
C	Cycles per tone (0 - 65535) number of cycles, duration.
P	Pause count (624 microseconds per count)
T	Tone number (0 - 4095) zero is highest frequency

The duration of a single tone may be approximately calculated using the number of cycles (C) and the following table:

T	ms/cycle
0	.624
1−5	.156*(T+2)
6−4095	.156*(2*T−4)

Multiply ms/cycle by number of cycles (C) to obtain ms per tone.

Constant time/tone >4 : 3000/(2T-4)

Except for tones 0, 1 and 2; pressing any key will stop the tone. A keypress will also stop the tone during pause. Either X or Y registers may also be specified in binary with this hexadecimal form:

```
Y = x x x x x x x R R R C C C C
X = x V V P P P P P P P P T T T
```

x = don't care
V = Volume Not

A few examples to try:

Y= 100	Y= 4,000	Y= 3,145,808	Y= 1,048,592	Y= 1,048,831	Y= 1,048,832
X= 30	X= 0	X= 131,072	X= 65,536	X= 1,044,481	X= 15,728,673

```
Y=         500 040
X= 10 F00 000 060 004
```

Increment or decrement the number of cycles (C) to disable buzz during pause time. Except for tone zero, even number of cycles makes buzz, odd number of cycles : no buzz. Flag 26 must be set to produce tone.

Because tones 0, 1 and 2 do not scan the keyboard, it is possible to get a 40 second tone that cannot be stopped by normal methods (ex: 65535 ENTER^ 0 TONEZ). It is possible in this case to stop the tone using the HP41C hard crash; hold down the Backarrow key and quickly press the ON key. This procedure may or may not turn off the calculator and may or may not result in buzz mode. Buzz mode may be stopped by executing any TONE or TONEZ or BEEP. While in buzz mode try this: XEQ ALPHA ON ALPHA.

LIFO functions

10

The LIFO (Last In First Out) functions use the same pointer as the ΣREG pointer. Thus care should be taken that the Sum Register functions and LIFO functions are not used simultaineously. The LIFO stack grows upwards in memory, that is; a push will increment the pointer and a pop will decrement the pointer. The LIFO pointer always resides at the top of the stack, a push will write to the register specified by the current pointer and a pop will read from the register below the register specified by the current pointer. ΣREG functions may be used momentarily in a subroutine.
Pointer manipulation functions:

ΣREG or ΣREG IND

Sets the ΣREG pointer to the register specified in main memory.

LIFOX

Sets the LIFO pointer the the absolute register specified in X (0-1023).

ΣREG?

Returns the location of the ΣREG in main memory (relative to absolute R00).
This function available on HP41CX.

LIFO?

Returns the value of the LIFO pointer (absolute register number).

$\boxed{\text{SIZE or PSIZE}}$

These functions modify the LIFO pointer, incrementing it by the amount SIZE is reduced or decrementing it by the amount SIZE is increased. PSIZE is an extended function (XF or CX).

Although LIFOX will allow setting the LIFO pointer to any absolute register number, the LIFO functions, as well as the ΣREG functions, have limits as to the registers which may be used. The ΣREG functions will only operate if the pointer is six registers or more below the top of main memory (absolute 512 on a CV/CX) and above the definition of R00. The LIFO functions will operate anywhere above absolute R00 or at absolute registers 65 to 189, assuming that these registers exist. The LIFO functions also recognize any register with all mantissa bits set as LIFO LIMIT and will not push into or pop from a register with such data.

LIFO functions: Z is X and Y, ST is STACK, F is FLAGS, and A is ALPHA.

$\boxed{\uparrow \text{X}}$ $\boxed{\uparrow \text{Z}}$ $\boxed{\uparrow \text{ST}}$ $\boxed{\uparrow \text{F}}$ $\boxed{\uparrow \text{A}}$

$\boxed{\text{POPX}}$ $\boxed{\text{POPZ}}$ $\boxed{\text{POPST}}$ $\boxed{\text{POPF}}$ $\boxed{\text{POPA}}$

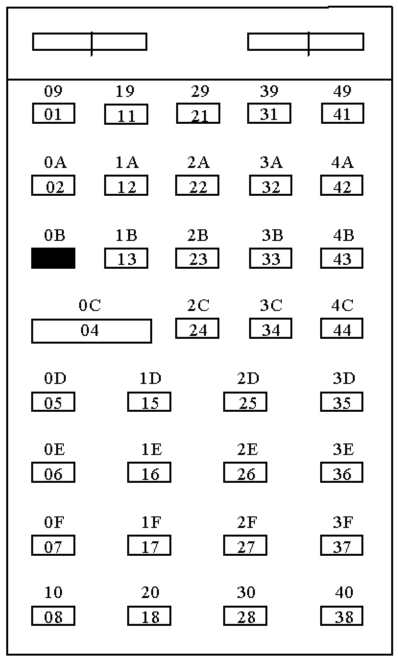
If the stack lift is disabled, POPX and POPZ do not cause a lift, eg, CLX POPZ does not modify Z and T registers. For multi-register push and pop functions, an LIFO LIMIT error leaves the stack or Alpha in an unknown state and the LIFO pointer is left in an unknown state. For POPA, if a DATA ERROR occurs the Alpha register has not been modified and the LIFO pointer has not been modified.

Not allowing usage of absolute registers 64, 190 and 191 is sort of a feeble attempt to protect the extended memory file system. If one wishes to use extended memory for the LIFO, carefull setup must be used to protect extended memory files, specifically, inserting limit register/s (all mantissa bits set) to prevent overflow/underflow. Also, one must remember that a re-size changes the LIFO pointer. A safe (maintains LIFO pointer) re-size sequence using PSIZE assuming new size in X: LIFO? X<>Y PSIZE X<>Y LIFOX.

With LIFO it is possible to write user code subroutines which simulate monadic functions, for example; do a push stack at entry, put the result in LASTX, then POPST and X<>L RTN. It is also possible to write interrupting alarms which actually do something, they can push the LASTX/stack/Alpha/flags at entry and recover them at exit. Note: the CLΣ function clears the six registers above the LIFO, it does not distroy stack data.

Finally, one must remember the basic rule for LIFO stack usage: whatever gets pushed MUST be poped! Otherwise we get what is known as a "memory leak" and eventual LIFO LIMIT error.

Key Assignment Function



A – KEY

Alpha to key assignment. This function creates up to eight key assignments for rom or mainframe functions. It is directed by a special binary string in Alpha. It also allows up to twelve keys to be cleared or any combination of clearing and assigning. Although A-KEY cannot create ram program global key assignments, it can clear these assignments. Each assignment sequence is three bytes whereas each clear sequence is two bytes. The first byte in each sequence is the keycode defined in the graphic above. To clear that key, the second byte is zero (00).

KK00	clear key KK, where KK is defined in the graphic above. Two bytes.
KKRRAA	assign RRAA to key KK. RR is not equal to zero and is equal to 04 for mainframe functions, 3 bytes.

To clear the top row of keys; put this hex code in Alpha using ABIN : 0100 1100 2100 3100 4100 (10 bytes). then execute A-KEY. Example; assign BST to the TAN key: 420407 (3 bytes).

Example; assign BLD to the LN key: 41A141 (41 is the LN keycode and A141 is the XROM code for BLD).
Example; Assign X<>Y to X<>Y and RDN to RDN and clear the XEQ key : 020471 120475 1300 (8 bytes).
Example; Assign the complex functions (shifted): 0DA179 0EA17A 0FA17B 10A17C (12 bytes).

If you need to know the hex code for a function, key it in as text into Alpha and execute A-41C, then BINHEX. For mainframe functions, use 04 for RR. Synthetic key assignments are interesting, for example, RCL 02 : 330422, unfortunately, synthetic key assignments do not preview display correctly. Some synthetic key assignments behave badly, even distructively.

This function runs at about 220 ms per assignment. This is not a "smart" function, it does not check the input for validity.

This rom also has a hidden function that packs the key assignment registers when the calculator is turned off.

These functions are intended to increase efficiency in running progams. Each requires a text line which Immediately followes the function in program memory. The text line is not loaded into Alpha, it is loaded into the stack, either X or X and Y. One or two stack lifts will occur, these functions do not respect the stack lift status. LDB and LDH do not notify the printer.

LDH

LoaD Hexadecimal text string which follows into X register. The binary equivalent of the text hexadecimal (characters 0-F) is pushed onto the stack. Use of this function is not as efficient as LDB, however the binary loaded into X is fully documented data when you review the program or print the program. Maximum 14 characters.

LDI

LoaD Integer into X register. The binary value of the string which follows is converted into an integer (less than 1E10) and pushed into X. This allows integers of large value to be generated rapidly and with minimum byte count. For example: to generate the integer 785,468,327 in the X register, the complete hex code including the LDI function is A1 67 F4 2E D1 4B A7. Key in 9 999 999 999 into X then execute BIN to see the maximum code that can be used for an integer. This function is byte count efficient for integers greater than four digits. Printer is notified that X has changed.

LDA

LoaD Alpha string. An Alpha string from one to six characters is loaded into the X register. If the string is seven to twelve characters long, both X and Y are loaded with strings. If a string does not follow the function in program memory, an Alpha Null string is loaded into X. This is usefull also for loading "safe" binary data into X, that is: it is a valid data type. Maximum 12 characters. Printer is notified.

LDB

LoaD Binary into X or X and Y. Loads a binary string directly from program memory into the X register or the X and Y registers. The fastest and simplest of the Load Immediate Functions. In order to load a normal number into X, the string must be seven characters long. In order to load normal numbers into X and Y the string must be 14 characters long. No conversions are performed, the text string byte image is simply transfered (pushed) into the stack. For strings which are not 7 or 14 characters long, the string is pushed into the stack as if it were an Alpha register. For example, a string of four characters is pushed into X right justified as binary NNN. If the string is nine characters long, the first two bytes will end up in

Y as right justified binary NNN. (NNN = Non Normalized Number).

Example: -5.768567895E-27 ENTER -8.423567895E-43 requires 33 bytes in program memory and takes 2 seconds.
LDB "FourteenCharString" 17 bytes and runs in 128 ms, here is the hex code including LDB :

A163 FD 95 76 85 67 89 59 73 98 42 35 67 89 59 57

This example could also be done efficiently using RCL N and RCL M, yet the above method is more efficient, faster and does not use the Alpha register. The load function loads characters into the stack faster than 41C does into Alpha.

A – 41C

Given text in the Alpha register which represents an HP41C user code line, this functions converts the text inplace to HP41C user code bytes. See BLD for syntax rules.
Handy if one does not have a byte table readily available.

A – END

Given HP41C user code bytes in the Alpha register, transfers those bytes just above the final .END..
No checking is done to ensure that the bytes in Alpha are valid HP41C user code. However, if the bytes represent a global line: LBL "ABC", or END : the line will be installed into the global chain. This installation into the global chain only takes place if the first line in Alpha is a global LBL or END (leading C0 byte).

Careless use of this function will probably result in MEMORY LOST.

Bytes are loaded at the first available location in program memory just above the .END., this allows fully compiled and fully packed user programs to be assembled. A register open will occur if needed. The .END. flags are always set to repack, an END 9 line (which will be fully packed) transfered will isolate the assembled program from the final .END.

END : The third byte of an END line is the status bits (76543210).

bit	definition	a normal newly created END has all bits clear.
0	marker	if a register open occurs, bits 0-3 are set.
1	decompile	if bit 1 is set, switch out of PRGM mode decompiles and clears bit 1.
2	repack	pack clears bit 2, pack will also clear bit 1 if bit 2 was set in any
3	marker	program, even any program above.
4		the marker bits do not seem to be used for anything.
5	.END.	final .END. at the bottom of program memory.
6		do not load an .END. , this will result in bad behavior.
7		bits 4-7 set indicate global LBL.

A-END does not modify the decompile flag, yet always sets the repack flag on the .END.. A-END sets the repack flag only because it is possible to load packable nulls with this function. A-END loads the bytes fully packed unless there are packable nulls within the byte string.

Pack will pack even if bit 1 and bit 2 are clear when a program above is flagged for repack. However, it will not decompile in this case, therefore a program flagged as fully packed must not have any packable nulls within it: compiled jumps may become invalid.

BLD does not modify the decompile bit when executed from within program mode and this is a violation of protocol. This method is required to allow assembling precomplied GTO and XEQ lines. When BLD is executed from within program mode it does not decompile before exit from program mode, however, if BLD is exited normally (via R/S) one is back in program mode and normal protocol is maintained.

Example programs

```
01 LBL "ABSRCL"
RTN?   GTO 00   "REG?"   PROMPT   FS? 21   ADV

08 LBL 00
CLA   XAI   48   3   ALENGXY   82   XAL   "□= "   ASTO Y   R-   BIN   PRSS   "FF 6C 55"   BINHEX
CLX   14   R-   X<>Y   ALENGXY   R-   XAL   FS? 21   XEQ 03   LASTX   DSE X   RTN?   RTN
FS? 21   OUTA   FS? 21   GTO 00   AVIEW   STOP   GTO 00

43 LBL 03
8   AROT   "□ "   4   AROT   "□ "   2   AROT   "□ "   X<>Y   AROT   "□ "   X<>Y   AROT   END          118 bytes
```

This program is handy for looking at absolute register data in hex. It works with or without a printer and may be called as a subroutine. The line following PRSS is three byte text (F3). The PRSS sequence is such that the program could be full stack traced, the sequence is SETXX STOY SETX 5 X<>Y so that the NNN ends up in Alpha and the PIL never has a chance to normalize it. If a printer is present, clear flag 21 to use the display interface. AROT is an XF function, if this function is not available, omit FS? 21 XEQ 03 and the subroutine. If the 82143A printer is being used, replace OUTA with PRA.

The ERAMCO Systems rom has a very usefull function named MOVE for rom-ram. Unfortunately, it requires keying hex digits into Alpha which can be painfull. The following program removes the pain.

```
01 LBL "MOVR"
CLA   DISPLAY   "bbbbeeeedddd"   TONE P   ABIN   BINHEX   MOVE   DISPLAY   END          37 bytes
```

As for MOVE, bbbb begining address, eeee ending address and dddd destination address. Press R/S to do another. The hex code for TONE P is 9F78.