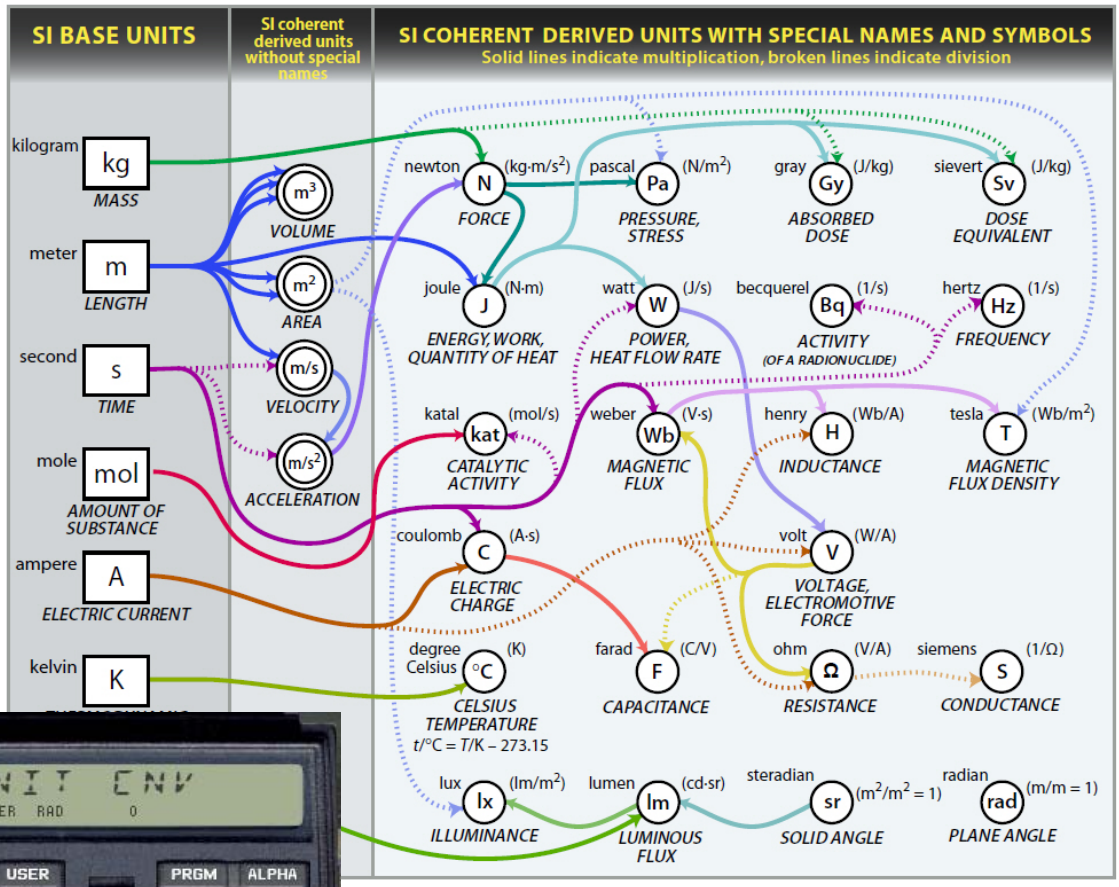
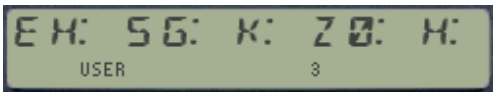
UMS-41 -- KLIB

A Constants Library for the Unit Conversion.





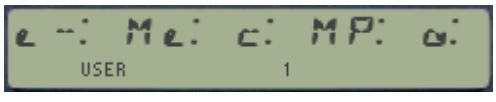
Written and prepared by:
Ángel Martin - June 2012

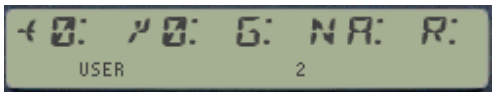
This compilation, revision A.2. 0.

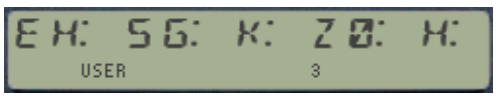
Table of contents.

Constants Library Table

#	Name	Description	Value	Units
1	**e-**	Electron Charge	-1,6021764 E-19	CB
2	**Me**	Electron Mass	9,1093821 E-31	KG
3	**c**	Speed of Light	2,9979245 E 08	M/S
4	**MP**	Proton's Mass	1,6726216 E-27	KG
5	**a**	Free-fall Acceleration	9,8066500 E 00	M/S2
6	**e0**	Vaccum Permitivity	8,8541878 E-12	FD/M
7	**u0**	Vacuum Permeability	1,2566370 E-06	N/A2
8	**G**	Gravity Constant	6,6742800 E-11	N*M2/KG
9	**NA**	Avogadro's Number	6,0221417 E 23	1/MOL
10	**R**	Gas Constant	8,314472150	J/K*MOL
11	**EH**	Hartree Energy	4,359748226 E-18	J
12	**SG**	Stefan-Boltzmann	5,6705119 E-8	W/M2*K4
13	**K**	Boltzmann's Constant	1,3806504 E-23	J/K
14	**Z0**	vacuum impedance	376,7303134	OHM
15	**H**	Planck's h constant	6,6260689 E-34	J*S
16	**F0**	magnetic flux quantum	2,0678336-15	WB
17	**a0**	Bohr Radius	5,2917720-11	M
18	**U**	atomic mass unit	1,6605387-27	KG
19	**G0**	conductance quantum	77,480916-06	1/OHM
20	**F**	Faraday constant	96.485,34	CB/MOL







1. Introduction.

This annex documents a new functionality added to the Unit Conversion Module: function **KLIB**, the Constants Library. It's meant to be an appendix to the main UMS-41 write-up.

In its original implementation (*) the module included 15 constants, each of them setup as an individual FAT entry and thus allowing individual access in RUN and PRGM modes. It however lacked a library catalog, offering a more convenient way to access them.

Besides the obvious advantage in usability, with this new implementation only one FAT entry is used – allowing for more functions to be added to the ROM. The use in PRGM mode is also possible by selecting the specific constant by its index (a number from 1 to 20), as independent program line following **KLIB**.

The following improvements have been made:

1. Added *5 more constants* to the library – for a total of 20. Not surprisingly quite a few of them are related to the EE field, showcasing both the UMS extensions and the author's background.

2. Constants are presented in the display as line-up menus, *with 4 groups of 5 entries each.*

3. In RUN mode, the selection is made using the top row keys, with the **SST** key used to navigate between the four "line-up" screens. Flag annunciators 1-4 indicate the current screen.

4. In PRGM mode the selection is made by adding an index number after KLIB, using the non-merged functions technique: two program lines in total for the complete information, not disrupting the stack.

5. The constant value is placed in the X register (stack is lifted), and its units are written in Alpha – ready for any unit conversion activity.

Because of the limited capabilities of the HP41 display, some of the symbols used to represent the constants differ from the standard notation. Slightly improved notation is possible using the lower case letters available in the "halfnut" character set but I have used the standard "fullnut" character set for compatibility with all 41 models (and the CL specially).

Currently **KLIB** is available in the POWER_CL module only, which (thanks to its bank-switched implementation) includes the complete Unit conversion functionality as well. **KLIB** will be added to the UNIT CONV module at a later time.

(*) *Refer to the main manual for information on the UNIT CONV module and the unit conversion approach and implementation techniques used by HP and the extensions made by the author.*

2. Constants Library.

What good is a Unit Conversion Module without a constants library? A pretty good one if you ask the author- but nevertheless a few universal constants are programmed into the module for completion sake – and now they're back with a vengeance

Despite the inherent limitations of the HP41 display and reduced character set, **KLIB** manages to implement a nice and convenient user interface. It follows the same model available in newer calculators, like the 33S and the 35S, so chances are that you're probably already familiar with it.

- Use the **SST** key to navigate to the next menu screen, from 1 to 4 repeated in sequential way. The flag appropriate indicator – 1,2,3,4 – will tell the current "screen" shown, so you'll know where you are.

- Use the **BackArrow** key to cancel out – no constant will be selected.

The constants included, their values and appropriate units in which they are expressed are listed in the following table:

#	Name	Description	Value	Units
1	**e-**	Electron Charge	-1,6021764 E-19	CB
2	**Me**	Electron Mass	9,1093821 E-31	KG
3	**c**	Speed of Light	2,9979245 E 08	M/S
4	**MP**	Proton's Mass	1,6726216 E-27	KG
5	**a**	Free-fall Acceleration	9,8066500 E 00	M/S2
6	**ε0**	Vaccum Permitivity	8,8541878 E-12	FD/M
7	**μ0**	Vacuum Permeability	1,2566370 E-06	N/A2
8	**G**	Gravity Constant	6,6742800 E-11	N*M2/KG
9	**NA**	Avogadro's Number	6,0221417 E 23	1/MOL
10	**R**	Gas Constant	8,314472150	J/K*MOL
11	**EH**	Hartree Energy	4,359748226 E-18	J
12	**SG**	Stefan-Boltzmann	5,6705119 E-8	W/M2*K4
13	**K**	Boltzmann's Constant	1,3806504 E-23	J/K
14	**Z0**	Vacuum impedance	376,7303134	OHM
15	**H**	Planck's constant	6,6260689 E-34	J*S
16	**F0**	Magnetic flux quantum	2,0678336-15	WB
17	**a0**	Bohr Radius	5,2917720-11	M
18	**U**	Atomic mass unit	1,6605387-27	KG
19	**G0**	Conductance quantum	77,480916-06	1/OHM
20	**F**	Faraday constant	96.485,34	CB/MOL

Note that ε0 uses the "left goose" character as a surrogate for epsilon.

Remember that not only the constant value is written in X, but also that *its units are returned into the alpha register* when the function is called – so you always know how they're expressed.

In program mode, the constant selection is made using the corresponding index as a program line right after the **KLIB** step. No menu will be offered, and therefore no hot keys either - completely automated.

3. Programming Highlights.

This section will be of interests to those with an inclination for "looking under the hood", as it describes the MCODE implementation of **KLIB**.

As it's always the case with user interface enhancements - and despite being such a modest library - there's a considerable amount of code required to make it work – all beyond the obvious programming of the constants values in ROM.

KLIB uses the partial data entry technique to offer the menu as a prompt text, then halts and waits for the user to press a key to "fill the prompt". Here are the main considerations that went into it:

- Since there are four different "prompts", there must be a way to switch to the next screen – implemented by pressing **SST**, as a "next screen" hot key.

- The five top keys are the only valid entries, so there must be a way to discard all others. This is relatively simple using CPU flag 4 (set when the two row keys are used) and a bit of logic to further restrict it to the A-E range, ignoring the F-J options. In fact a different approach was followed, using a look-up table instead due to the reasons explained below.

- A little trickier is to distinguish the same key amongst the four different screens possible. This is implemented using CPU flags 10 & 11, with a simple table for all four cases.

There are 20 valid inputs to the prompts, which are programmed into a look-up table in the ROM. Each of the table entries has two bytes, as follows:

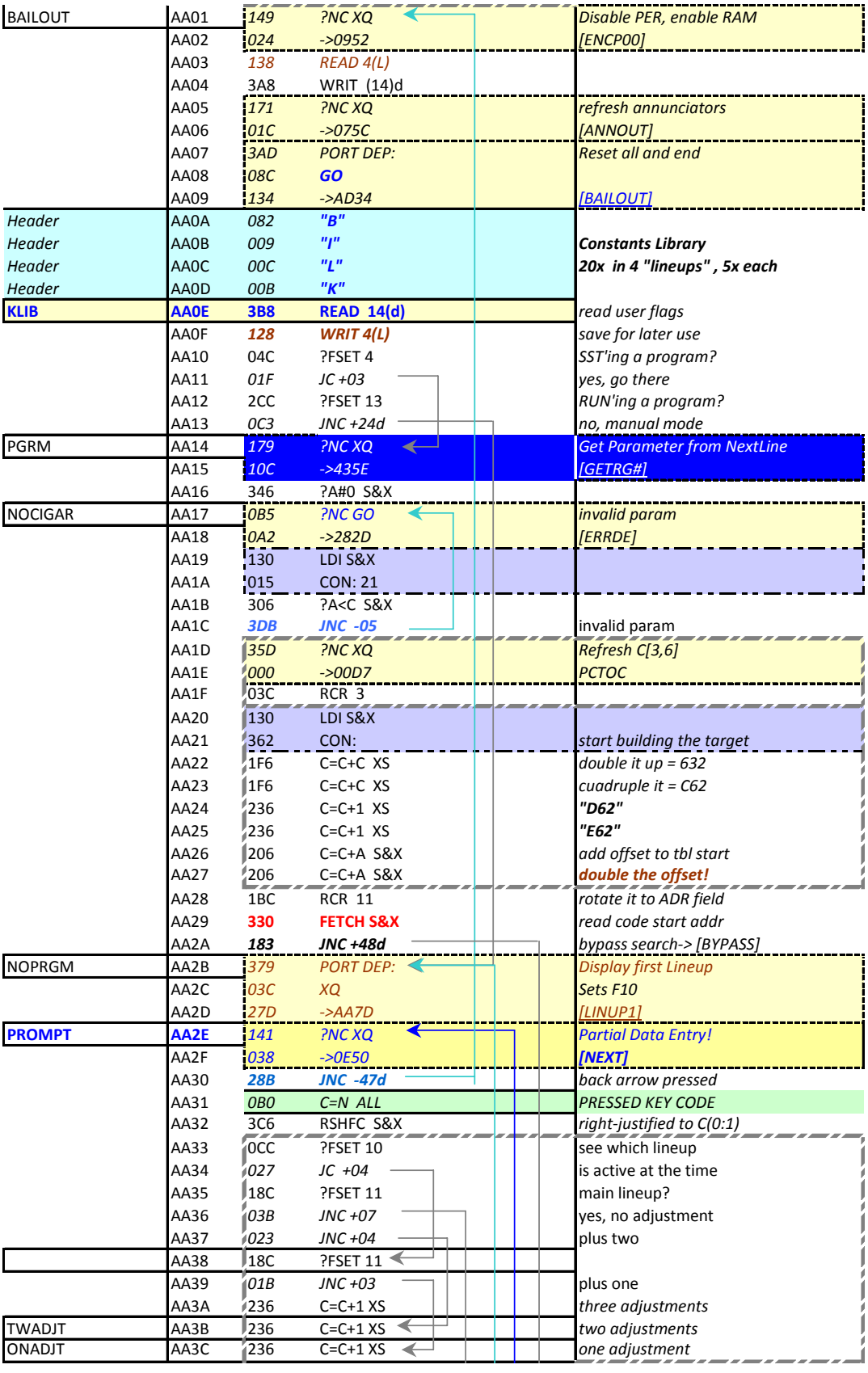
- Byte-1 holds the selected screen/key value. Here we've used the XS digit as screen indicator, while the rest of the S&X field holds the key code.

- Byte-2 holds the address in ROM where the constant value is programmed – so a simple GOTOADR will transfer the execution to the proper place.

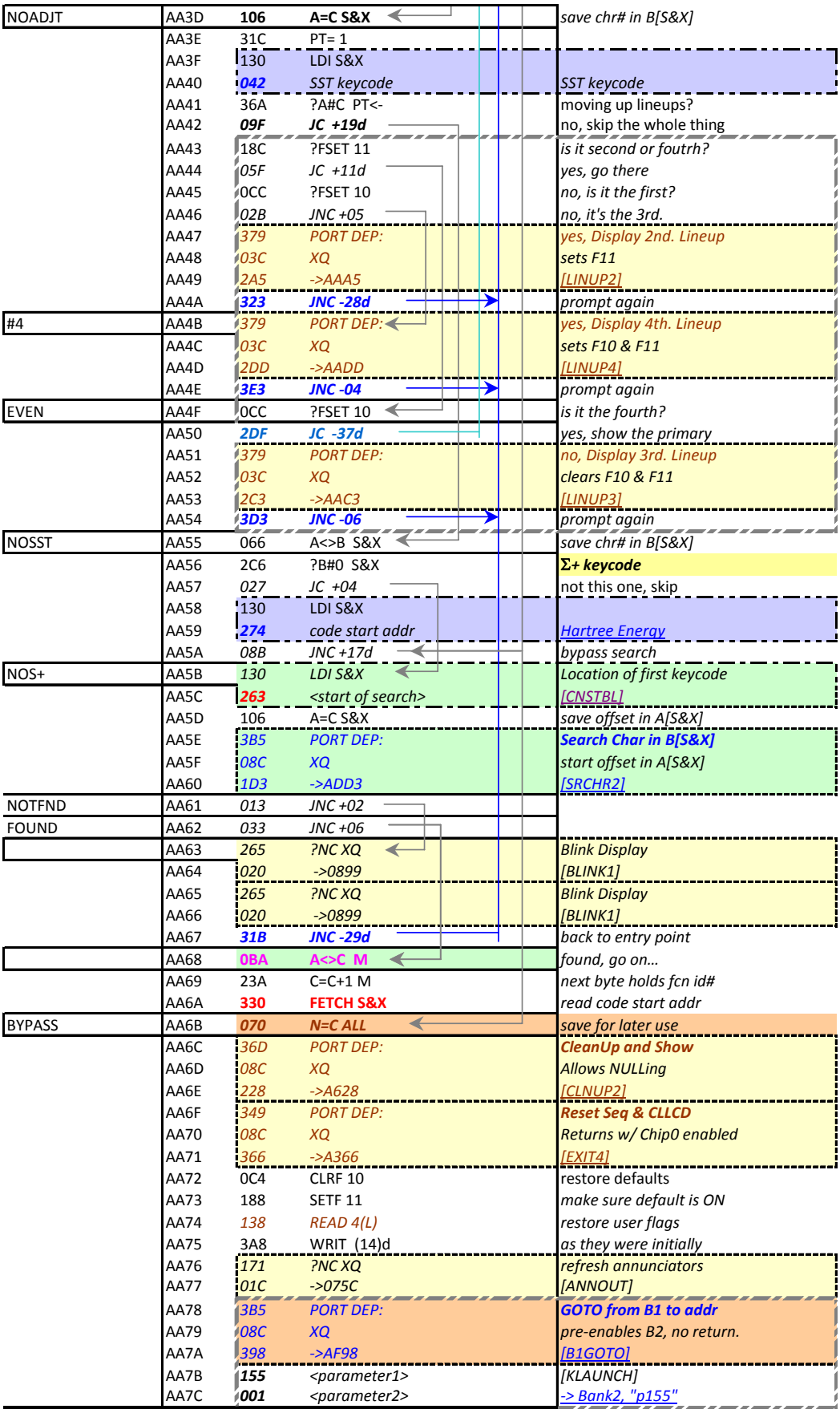
So the workflow is rather simple: when a valid key (A-F, BA, SST) is pressed the routine will check the input keycode against the byte-1 values on the table. If no match is found, the prompt will be maintained. If a match is found the routine will reset things to the normal state and the execution will be transferred to the address in byte-2.

Four subroutines are used to write the prompts into the display, triggered by **SST** and controlled the status of CPU flags 10/11. They use a combination of direct write instructions, and calls to the OS [MESSL] routine – which doesn't handle special characters.

So without further ado, let's get into the MCODE g(l)ory details, starting with the main code section of KLIB, followed by the line-ups building routines, and ending with the vital table search and table structure documentation.



BAILOUT	AA01	*149*	*?NC XQ*	*Disable PER, enable RAM*
	AA02	*024*	*->0952*	*[ENCP00]*
	AA03	*138*	*READ 4(L)*	
	AA04	3A8	WRIT (14)d	
	AA05	*171*	*?NC XQ*	*refresh annunciators*
	AA06	*01C*	*->075C*	*[ANNOUT]*
	AA07	*3AD*	*PORT DEP:*	*Reset all and end*
	AA08	*08C*	***GO***	
	AA09	*134*	*->AD34*	*[BAILOUT]*
Header	AA0A	*082*	***"B"***	
Header	AA0B	*009*	***"I"***	***Constants Library***
Header	AA0C	*00C*	***"L"***	***20x in 4 "lineups" , 5x each***
Header	AA0D	*00B*	***"K"***	
KLIB	**AA0E**	**3B8**	**READ 14(d)**	*read user flags*
	AA0F	***128***	***WRIT 4(L)***	*save for later use*
	AA10	04C	?FSET 4	*SST'ing a program?*
	AA11	*01F*	*JC +03*	*yes, go there*
	AA12	2CC	?FSET 13	*RUN'ing a program?*
	AA13	*0C3*	*JNC +24d*	*no, manual mode*
PGRM	AA14	*179*	*?NC XQ*	*Get Parameter from NextLine*
	AA15	*10C*	*->435E*	*[GETRG#]*
	AA16	346	?A#0 S&X	
NOCIGAR	AA17	*0B5*	*?NC GO*	*invalid param*
	AA18	*0A2*	*->282D*	*[ERRDE]*
	AA19	130	LDI S&X	
	AA1A	015	CON: 21	
	AA1B	306	?A<C S&X	
	AA1C	*3DB*	*JNC -05*	invalid param
	AA1D	*35D*	*?NC XQ*	*Refresh C[3,6]*
	AA1E	*000*	*->00D7*	*PCTOC*
	AA1F	03C	RCR 3	
	AA20	130	LDI S&X	
	AA21	362	CON:	*start building the target*
	AA22	1F6	C=C+C XS	*double it up = 632*
	AA23	1F6	C=C+C XS	*cuadruple it = C62*
	AA24	236	C=C+1 XS	***"D62"***
	AA25	236	C=C+1 XS	***"E62"***
	AA26	206	C=C+A S&X	*add offset to tbl start*
	AA27	206	C=C+A S&X	***double the offset!***
	AA28	1BC	RCR 11	*rotate it to ADR field*
	AA29	**330**	**FETCH S&X**	*read code start addr*
	AA2A	***183***	***JNC +48d***	*bypass search-> [BYPASS]*
NOPRGM	AA2B	*379*	*PORT DEP:*	*Display first Lineup*
	AA2C	*03C*	*XQ*	*Sets F10*
	AA2D	*27D*	*->AA7D*	*[LINUP1]*
PROMPT	**AA2E**	*141*	*?NC XQ*	*Partial Data Entry!*
	AA2F	*038*	*->0E50*	***[NEXT]***
	AA30	***28B***	***JNC -47d***	*back arrow pressed*
	AA31	*0B0*	*C=N ALL*	*PRESSED KEY CODE*
	AA32	3C6	RSHFC S&X	*right-justified to C(0:1)*
	AA33	0CC	?FSET 10	see which lineup
	AA34	*027*	*JC +04*	is active at the time
	AA35	18C	?FSET 11	main lineup?
	AA36	*03B*	*JNC +07*	yes, no adjustment
	AA37	*023*	*JNC +04*	plus two
	AA38	18C	?FSET 11	
	AA39	*01B*	*JNC +03*	plus one
	AA3A	236	C=C+1 XS	*three adjustments*
TWADJT	AA3B	236	C=C+1 XS	*two adjustments*
ONADJT	AA3C	236	C=C+1 XS	*one adjustment*



NOADJT	AA3D	**106**	**A=C S&X**	*save chr# in B[S&X]*
	AA3E	31C	PT= 1	
	AA3F	130	LDI S&X	
	AA40	***042***	*SST keycode*	*SST keycode*
	AA41	36A	?A#C PT<-	moving up lineups?
	AA42	***09F***	***JC +19d***	no, skip the whole thing
	AA43	18C	?FSET 11	*is it second or foutrh?*
	AA44	*05F*	*JC +11d*	*yes, go there*
	AA45	0CC	?FSET 10	*no, is it the first?*
	AA46	*02B*	*JNC +05*	*no, it's the 3rd.*
	AA47	*379*	*PORT DEP:*	*yes, Display 2nd. Lineup*
	AA48	*03C*	*XQ*	*sets F11*
	AA49	*2A5*	*->AAA5*	*[LINUP2]*
	AA4A	***323***	***JNC -28d***	*prompt again*
#4	AA4B	*379*	*PORT DEP:*	*yes, Display 4th. Lineup*
	AA4C	*03C*	*XQ*	*sets F10 & F11*
	AA4D	*2DD*	*->AADD*	*[LINUP4]*
	AA4E	***3E3***	***JNC -04***	*prompt again*
EVEN	AA4F	0CC	?FSET 10	*is it the fourth?*
	AA50	***2DF***	***JC -37d***	*yes, show the primary*
	AA51	*379*	*PORT DEP:*	*no, Display 3rd. Lineup*
	AA52	*03C*	*XQ*	*clears F10 & F11*
	AA53	*2C3*	*->AAC3*	*[LINUP3]*
	AA54	***3D3***	***JNC -06***	*prompt again*
NOSST	AA55	066	A<>B S&X	*save chr# in B[S&X]*
	AA56	2C6	?B#0 S&X	***Σ+ keycode***
	AA57	*027*	*JC +04*	not this one, skip
	AA58	130	LDI S&X	
	AA59	***274***	*code start addr*	*Hartree Energy*
	AA5A	*08B*	*JNC +17d*	*bypass search*
NOS+	AA5B	*130*	*LDI S&X*	*Location of first keycode*
	AA5C	***263***	*<start of search>*	*[CNSTBL]*
	AA5D	106	A=C S&X	*save offset in A[S&X]*
	AA5E	*3B5*	*PORT DEP:*	***Search Char in B[S&X]***
	AA5F	*08C*	*XQ*	*start offset in A[S&X]*
	AA60	*1D3*	*->ADD3*	*[SRCHR2]*
NOTFND	AA61	*013*	*JNC +02*	
FOUND	AA62	*033*	*JNC +06*	
	AA63	*265*	*?NC XQ*	*Blink Display*
	AA64	*020*	*->0899*	*[BLINK1]*
	AA65	*265*	*?NC XQ*	*Blink Display*
	AA66	*020*	*->0899*	*[BLINK1]*
	AA67	***31B***	***JNC -29d***	*back to entry point*
	AA68	**0BA**	**A<>C M**	*found, go on…*
	AA69	23A	C=C+1 M	*next byte holds fcn id#*
	AA6A	**330**	**FETCH S&X**	*read code start addr*
BYPASS	AA6B	***070***	***N=C ALL***	*save for later use*
	AA6C	*36D*	*PORT DEP:*	***CleanUp and Show***
	AA6D	*08C*	*XQ*	*Allows NULLing*
	AA6E	*228*	*->A628*	*[CLNUP2]*
	AA6F	*349*	*PORT DEP:*	***Reset Seq & CLLCD***
	AA70	*08C*	*XQ*	*Returns w/ Chip0 enabled*
	AA71	*366*	*->A366*	*[EXIT4]*
	AA72	0C4	CLRF 10	restore defaults
	AA73	188	SETF 11	*make sure default is ON*
	AA74	*138*	*READ 4(L)*	*restore user flags*
	AA75	3A8	WRIT (14)d	*as they were initially*
	AA76	*171*	*?NC XQ*	*refresh annunciators*
	AA77	*01C*	*->075C*	*[ANNOUT]*
	AA78	*3B5*	*PORT DEP:*	***GOTO from B1 to addr***
	AA79	*08C*	*XQ*	*pre-enables B2, no return.*
	AA7A	*398*	*->AF98*	*[B1GOTO]*
	AA7B	***155***	*<parameter1>*	*[KLAUNCH]*
	AA7C	***001***	*<parameter2>*	*-> Bank2, "p155"*

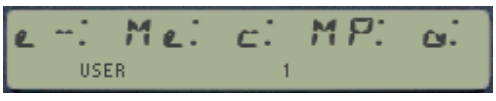
No doubt you've noticed the unusual way to transfer the execution to [KLAUNCH] – which is located in the second bank and therefore needs special handling. Its address is in the parameter1 and parameter2 bytes, used by [B1GOTO] to transfer the execution ***after*** enabling bank2.

Next come the four "line-ups" building routines, nothing special in here beyond write instructions and calls to [MESSL] OS routine.

LINUP1	AA7D	0C8	SETF 10	*next one is #2*
	AA7E	184	CLRF 11	
	AA7F	*149*	*?NC XQ*	*Disable PER, enable RAM*
	AA80	*024*	*->0952*	*[ENCP00]*
	AA81	3B8	READ 14(d)	*read flags*
	AA82	2DC	PT= 13	
	AA83	110	LD@PT- 4	*sets UF1*
	AA84	010	LD@PT- 0	
	AA85	3A8	WRIT (14)d	*re-write to flags register*
	AA86	*171*	*?NC XQ*	*refresh annunciators*
	AA87	*01C*	*->075C*	*[ANNOUT]*
	AA88	*3C1*	*?NC XQ*	*clear & enable LCD*
	AA89	*0B0*	*->2CF0*	*[CLLCDE]*
	AA8A	130	LDI S&X	
	AA8B	**105**	*"e"*	
	AA8C	***3E8***	***WRIT 15(e)***	*9-bit LCD write*
	AA8D	*3BD*	*?NC XQ*	
	AA8E	*01C*	*->07EF*	*[MESSL]*
	AA8F	*0AD*	*"-:"*	
	AA90	*020*	*" "*	
	AA91	*20D*	***"M"***	
	AA92	130	LDI S&X	
	AA93	**185**	*"e:"*	
	AA94	***3E8***	***WRIT 15(e)***	*9-bit LCD write*
	AA95	*3BD*	*?NC XQ*	
	AA96	*01C*	*->07EF*	*[MESSL]*
	AA97	*220*	*" "*	
	AA98	130	LDI S&X	
	AA99	**183**	***"c:"***	
	AA9A	***3E8***	***WRIT 15(e)***	*9-bit LCD write*
	AA9B	*3BD*	*?NC XQ*	
	AA9C	*01C*	*->07EF*	*[MESSL]*
	AA9D	*020*	*" "*	
	AA9E	*00D*	***"M"***	
	AA9F	*090*	***"P:"***	
	AAA0	*220*	*" "*	
	AAA1	130	LDI S&X	
	AAA2	**181**	***"a:"***	
	AAA3	***3E8***	***WRIT 15(e)***	*9-bit LCD write*
	AAA4	3E0	RTN	

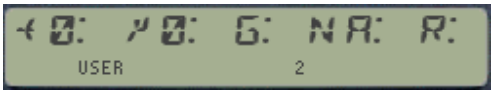
[LINUP1] is used at the beginning of the execution, therefore is responsible for saving the current user flags into register 4(L), so that the correct flag configuration will be restored after **KLIB** ends. In general each one of the line-up routines must do the flag handling so that the keycode will get properly marked with the screen digit signaled in the XS nibble.

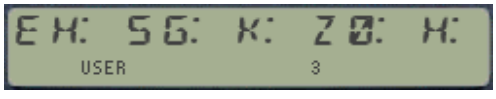
This is the output generated by [LINUP1]:



Subroutines [LINEUP2] and [LINEUP3] are shown next, as well as their outputs:

LINUP2	**AAA5**	**188**	**SETF 11**	next one is #3
	AAA6	0C4	CLRF 10	
	AAA7	*149*	*?NC XQ*	*Disable PER, enable RAM*
	AAA8	*024*	*->0952*	*[ENCP00]*
	AAA9	3B8	READ 14(d)	*read flags*
	AAAA	2DC	PT= 13	
	AAAB	090	LD@PT- 2	*sets UF2*
	AAAC	010	LD@PT- 0	
	AAAD	3A8	WRIT (14)d	*re-write to flags register*
	AAAE	*171*	*?NC XQ*	*refresh annunciators*
	AAAF	*01C*	*->075C*	*[ANNOUT]*
	AAB0	*215*	*?NC XQ*	*Build Msg - all cases*
	AAB1	*0FC*	*->3F85*	*[APRMSG2]*
	AAB2	*02C*	***" ε "***	
	AAB3	*0B0*	***"0:"***	
	AAB4	*220*	***" "***	
	AAB5	130	LDI S&X	
	AAB6	**10C**	" μ"	
	AAB7	***3E8***	***WRIT 15(e)***	*9-bit LCD write*
	AAB8	*3BD*	*?NC XQ*	
	AAB9	*01C*	*->07EF*	*[MESSL]*
	AABA	*0B0*	***"0:"***	*Lineup #2*
	AABB	*020*	***" "***	
	AABC	*087*	***"G:"***	
	AABD	*020*	***" "***	
	AABE	*00E*	***"N"***	
	AABF	*081*	***"A:"***	
	AAC0	*020*	***" "***	
	AAC1	*292*	***"R:"***	
	AAC2	3E0	RTN	
LINUP3	**AAC3**	**0C4**	**CLRF 10**	*next one is #4*
	AAC4	184	CLRF 11	
	AAC5	*149*	*?NC XQ*	*Disable PER, enable RAM*
	AAC6	*024*	*->0952*	*[ENCP00]*
	AAC7	3B8	READ 14(d)	*read flags*
	AAC8	2DC	PT= 13	
	AAC9	050	LD@PT- 1	*sets UF3*
	AACA	010	LD@PT- 0	
	AACB	3A8	WRIT (14)d	*re-write to flags register*
	AACC	*171*	*?NC XQ*	*refresh annunciators*
	AACD	*01C*	*->075C*	*[ANNOUT]*
	AACE	*215*	*?NC XQ*	*Build Msg - all cases*
	AACF	*0FC*	*->3F85*	*[APRMSG2]*
	AAD0	*005*	***"E"***	
	AAD1	*088*	***"H:"***	
	AAD2	*020*	***" "***	
	AAD3	*013*	***"S"***	
	AAD4	*087*	***"G:"***	*Lineup #3*
	AAD5	*020*	***" "***	
	AAD6	*08B*	***"K:"***	
	AAD7	*020*	***" "***	
	AAD8	*01A*	***"Z"***	
	AAD9	*0B0*	***"O:"***	
	AADA	*020*	***" "***	
	AADB	*288*	***"H:"***	
	AADC	3E0	RTN	





Note that the previous subroutines end with a RTN instruction after the call to [MESSL]. This shouldn't be strictly required, but it's just there as additional assurance.

Note also the calls to [APRMSG2] in the CX ROM, a shortcut that includes selecting the display followed by a call to [MESSL]. This introduces an obvious dependency to the CX OS, but it's far from being the only time where this is used – so not a new one.

The last one of the four line-ups is created using [LINUP4] shown below:

LINUP4	AADD	0C8	SETF 10	*next one is #1*
	AADE	188	SETF 11	
	AADF	*149*	*?NC XQ*	*Disable PER, enable RAM*
	AAE0	*024*	*->0952*	*[ENCP00]*
	AAE1	3B8	READ 14(d)	*read flags*
	AAE2	2DC	PT= 13	
	AAE3	010	LD@PT- 0	
	AAE4	210	LD@PT- 8	*sets UF4*
	AAE5	3A8	WRIT (14)d	*re-write to flags register*
	AAE6	*171*	*?NC XQ*	*refresh annunciators*
	AAE7	*01C*	*->075C*	*[ANNOUT]*
	AAE8	*215*	*?NC XQ*	*Build Msg - all cases*
	AAE9	*0FC*	*->3F85*	*[APRMSG2]*
	AAEA	*006*	***"F"***	
	AAEB	*0B0*	***"0:"***	
	AAEC	*220*	***" "***	
	AAED	130	LDI S&X	
	AAEE	**101**	***"a"***	
	AAEF	***3E8***	***WRIT 15(e)***	*9-bit LCD write*
	AAF0	*3BD*	*?NC XQ*	
	AAF1	*01C*	*->07EF*	*[MESSL]*
	AAF2	*0B0*	***"0:***	
	AAF3	*020*	***" "***	
	AAF4	*095*	***"U:"***	
	AAF5	*020*	***" "***	
	AAF6	*007*	***"G"***	
	AAF7	*0B0*	***"0:"***	
	AAF8	*020*	***" "***	
	AAF9	*286*	***"F:"***	
	AAFA	3E0	RTN	

And the output it creates (flag 4 indicator is ON):

At the core of KLIB is the table search routine, also used all throughout the POWER_CL module.

SRCHR2	ADD3	04E	C=0 ALL	*Expects chr# in B[S&X]*
	ADD4	*35D*	*?NC XQ*	
	ADD5	*000*	*->00D7*	*[PCTOC]*
	ADD6	03C	RCR 3	*get page number*
	ADD7	130	LDI S&X	
	ADD8	300	CON:	
	ADD9	1F6	C=C+C XS	*double it up = 600*
	ADDA	1F6	C=C+C XS	*cuadruple it = C00*
	ADDB	206	C=C+A S&X	***add offset to tbl start***
	ADDC	1BC	RCR 11	*put it in C(6:3)*
	ADDD	0BA	A<>C M	*put it in A(6:3)*
	ADDE	**066**	**A<>B S&X**	*put reference in A[S&X]*
GETCOD	ADDF	0BA	A<>C M	*bring adr to C[M]*
	ADE0	11A	A=C M	*keep it in A[M]*
	ADE1	330	FETCH S&X	*read KEYCODE*
	ADE2	2E6	?C#0 S&X	*value non-zero?*
	ADE3	**3A0**	**?NC RTN**	*NO, TERMINATE HERE!*
	ADE4	366	?A#C S&X	*are they different?*
	ADE5	*023*	*JNC +04*	*no, -> [FOUND]*
	ADE6	17A	A=A+1 M	*add offset until*
	ADE7	17A	A=A+1 M	*next addr field*
	ADE8	*3BB*	*JNC -09*	*loop back*
	ADE9	1B0	POPADR	
	ADEA	23A	C=C+1 M	
	ADEB	170	PUSHADR	*increase RTN adr*
	ADEC	**3E0**	**RTN**	

The table location is passed to [SRCH2] in A[S&X], and the character code in B[S&X]. The search will compare all of the byte-1 values against the sought-for code, ending when a match is found or when the table end is reached (zero value). The return address is incremented by one if a match was found.

Upon its return to the main code, **KLIB** will FETCH the address location from byte-2 on the table, which conveniently stored in N will be used by [KLAUNCH] from bank2 in a GOTOADR instruction to call the code snippet that will load the constant value in X, the units in ALPHA, and graciously exit to the OS after re-enabling bank-1.

Finally, the table structure and values are shown below:

CNSTBL	**AE63**	100	**Σ+,** 2nd. lineup	
	AE64	1F5	*electron charge*	*A176*
	AE65	110	**1/X, 2nd. Lineup**	
	AE66	307	*electron mass*	*A1A9*
	AE67	120	**SQRT, 2nd. Lineup**	
ction start	AE68	177	*speed of light*	*A15E*
	AE69	130	**LOG, 2nd. Lineup**	
	AE6A	323	*proton mass*	*A1C2*
	AE6B	140	**LN, 2nd. Lineup**	
	AE6C	1A8	*free-fall acceleration*	*A215*
	AE6D	200	**Σ+, 3**nd. lineup	
	AE6E	23F	*Vacuum Permitivity*	*A1F9*
	AE6F	210	**1/X, 3rd. Lineup**	
	AE70	227	*vacuum permeability*	*A1E1*
	AE71	220	**SQRT, 3nd. Lineup**	
	AE72	20E	*gravity constant*	*A18E*
	AE73	230	**LOG, 3rd. Lineup**	
	AE74	15D	*Avogardo's Nr.*	*A143*
	AE75	240	**LN, 3rd. Lineup**	
	AE76	1BC	*R Gas constant*	*A229*
	AE77	*013*	*non-existent key (!)*	*needed for PRGM case*
	AE78	274	*Hartree Energy*	*A267*
	AE79	010	**1/X** keycode	
	AE7A	259	*Stefan-Boltmann*	*A24F*
	AE7B	040	**LN** keycode	
	AE7C	28B	*Plank constant*	*A283*
	AE7D	030	**LOG** keycode	
	AE7E	18F	*vacuum impedance*	*A2FA*
	AE7F	020	**SQRT** keycode	
	AE80	2A4	*Boltzmann ct*	*A29C*
	AE81	300	**Σ+,** 4th. lineup	
	AE82	2D4	magnetic flux quantum	*A2CC*
	AE83	310	**1/X, 4th. Lineup**	
	AE84	2BD	Bohr radius	*A2B5*
	AE85	320	**SQRT, 4th. Lineup**	
	AE86	333	atomic mass unit	*A1D2*
	AE87	330	**LOG, 4th. Lineup**	
	AE88	2EC	conductance quantum	*A2E4*
	AE89	340	**LN, 4th Lineup**	
	AE8A	1D6	Faraday constant	*A314*
	AE8B	*000*	*end of table*	

And the rest, as they say, it's just history.

That's all folks; hopefully this brief write-up has been interesting to you. I had a lot of fun programming it and learned a few tricks along the way, I hope you enjoy it as much as I did.