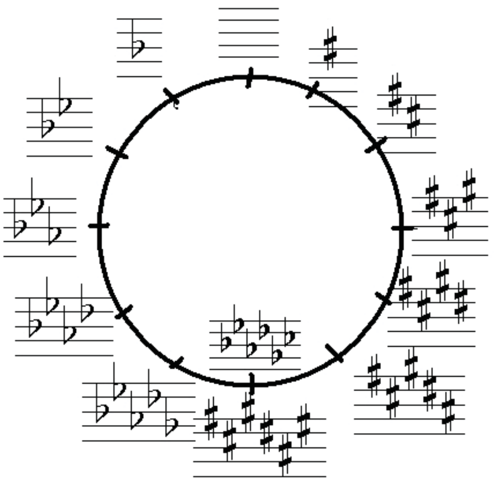
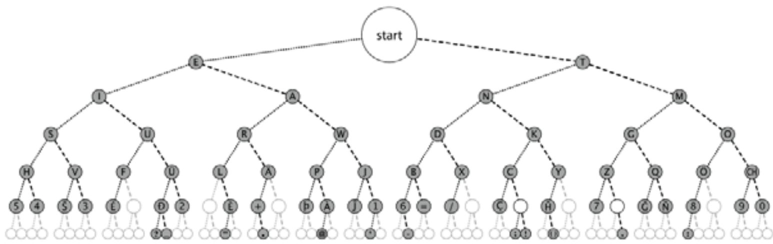
SUDOKU & Sound

Module for the HP-41



Mini-manual and QRG



Programmed by Jean-Marc Baillard & Ángel Martin

Revision 1C - December 2011

This compilation: revision A.1.1.

Screen captures taken from V41, Windows-based emulator developed by Warren Furlow.
See: http:\\www.hp41.org

Intro.

I've never done a Sudoku - somehow always found writing MCODE to be a more relaxing activity, being such a geek. I didn't even know the rules before working on this module, so perhaps you'd be thinking I'm hardly qualify to write Sudoku solving software - and you're right. We have to thank Jean-Marc Baillard for the Sudoku solvers and Grid programs, around which I built the rest of the routines for a more convenient usage and easier UI.

Whit this said, this is supposed to be a fun module even if by its nature Sudokus are brainy puzzles. There are two main sections, the "- SUDOKU" and the "-SOUND F/X" - loosely grouping sound-related functions and programs – plus two more about Magic Squares, slightly related to the same theme. These basically revisit some old concepts and even add a bit of tune playing to celebrate your next July 4th. with style - an indulgence with a beeper twist.

So here it is, straight from the "Martin-Baillard" module facture - hope you enjoy it!

#	Function	Description	Input	Output	Author
1	-SUDOKU	*Section Header*	*none*	*shows "Loading..."*	*Ángel Martin*
2	^SROW	Enter Row	row# in X	inputs string of values	*Ángel Martin*
3	ADEL	Alpha Back Space	string in Alpha	deletes last chr	*Ken Emery*
4	AINT	Append Integer	number in X	appends integer part	*Frits Ferwerda*
5	ANUMDL	ANUM w/ Deletion	string in Alpha	number in X / deletes chrs	*HP Co.*
6	E3/E+	1,00x	number in X	index preparation	*Ángel Martin*
7	GRID	Creates Sudoku GRID	type # blanks in X	Grid ready in R1-R9	*JM Baillard*
8	SDK	Fast Solver	Sudoku in R10-R19	Sudoku solved in R1-R9	*JM Baillard*
9	*"SDK0"*	Example #0	none	resolves blank Sudoku	*Ángel Martin*
10	*"SDK1"*	Example #1	none	resolves Sudoku #1	*Ángel Martin*
11	*"SDK2"*	Example #2	none	resolves Sudoku #2	*Ángel Martin*
12	SDKIN	Data input	none	Loads Data in R1-R81	*Ángel Martin*
13	SDKOUT	Data Output	Data in R1-R81	Loads Sudoku in R1-R9	*Ángel Martin*
14	*"SLSDK"*	Slow Solver	Data in R1-R81	Solves Data in R1-R81	*JM Baillard*
15	*"SUD"*	Housekeeping	Sudoku in R1-R9	Solves Sudoku in R1-R9	*Ángel Martin*
16	*"SUDOKU"*	Main Program	Under program control	Main program	*Ángel Martin*
17	*"SUDRPN"*	RPN Program	Under program control	Main program #2	*Ángel Martin*
18	*"SUDVIEW"*	View Sudoku	Sudoku in R1-R9	Views Sudoku in R1-R9	*Ángel Martin*
19	ΣDIG	Digit Sum	number in X	sum in X	*Ángel Martin*
20	-SOUND FX	*Section Header*	*none*	*Shows "Solving..."*	*Ángel Martin*
21	BUZZON	Set Buzzer Mode	none	sets busser mode	*Andreas Meyer*
22	CLAXON	Claxon sound	none	makes sound	*Mark Power*
23	NOBUZZ	Clear Buzzer Mode	none	clears buzzer mode	*Ángel Martin*
24	RASP	Rasp sound	none	makes sound	*Mark Power*
25	RNG	Random# w/ Timer	none	random number in X	*JM Baillard*
26	TONEXY	Configurable TONE	Length in X, Freq. in Y	plays tone	*JM Baillard*
27	*"C-N"*	Chords to Notes	Chord in Alpha	displays notes	*JM Baillard*
28	*"N-C"*	Notes to Chords	Notes in Alpha	displays chord name	*JM Baillard*
29	*"MORSE"*	Morse Code	Text in Alpha	plays Morse	*JM Baillard*
30	*"SSB"*	Star-spangled Banner	none	plays tune	*JM Baillard*
31	LEFT	Shifts display left	none	display shifted	*Nelson Crowle*
32	GOOSE	Puts Left Goose in Display	none	sets msg flag	*Nelson Crowle*
33	*"MAGIC"*	Magic Squares	prompts for order (odd)	F00 set stores values	*JM Baillard*
34	*"PANMG"*	Pan-Magic Squares	prompts for order	F00 set stores values	*JM Baillard*

Two main programs – *"SUDOKU"* and "SUDRPN".

The Sudoku module really is arranged around these two programs. The data entry and output routines are either integrated into them, or called as outside available functions in the ROM. The solving components are SDK and **SLSDK** (*SLow SDK*) respectively, the first one being a MCODE function is about 180 times faster than the second.

The structure of these programs can be represented by the following block diagram:

Data Entry → Processing → Data Output

With the table below showing the different functions used:

Program	*Data Entry*	*Processing*	*Data Output*
SUDOKU	**^SROW**	**SUD** (and **SDK**)	**SUDVIEW**
SUDRPN	**SDKIN**	**SLSDK**	**SDKOUT**

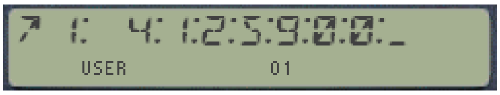
I. Grid data entry: the ^SROW function.

One of the nice aspects of the module is the fast and convenient way to enter the Sudoku grid data. The traditional approach would require prompting for each individual element, for a total of 81 – clearly a tiring and inefficient system.

Considering that the Sudoku elements are single digits (0 to 9), there must be a better way to go about the data entry process – and there is: meet the ^SROW function, which uses Alpha as vehicle for the digit input *in blocks of multiple elements at once*, (therefore arranged in rows) all with a special keyboard enabled for the task, as follows:

- *Numeric keypad, for 0-9 as element value*
- *Back arrow to delete previous entry or cancel out*
- *R/S, to terminate the entry sequence*
- *ON turns the calculator off*

The figures below show entry of the elements in rows #1 and #3 mid-way. Note how digits are separated by a colon, delimiting each individual digit. The left of the display has an input arrow plus a number, signaling the "row" being edited.



The editing process may be terminated at any time. Elements are being stored in Alpha, from where they'll be retrieved by the data input routine in a loop using function ANUMDL – converting the alpha string into a valid number and storing it in the appropriate data register. Missing values (if fewer than 9) will be replaced with zeroes, and excess elements (if more than 9) will be ignored.

II. Solving the puzzle: SDK and SLDK.

At the core of the Sudoku programs are the number-crunching engines, the real heart(s) which actually resolve the puzzles and write the results for the data output routines to pick-up. Both are written by Jean-Marc Baillard; see the corresponding pages on the appendix section of this mini-manual.

Execution times can be very long, therefore it's most recommended to use **TURBO50** on the 41CL – or a PC-emulator (like V41 running in turbo mode). Note also that some grids won't have a valid solution, which will be displayed as *"DATA ERROR"* or *"NO SOLUTION"-* both really meaning: "sorry, no cigar".

The best way to test the functionality is by solving the canned examples included in the module; ***"SDK0"***, ***"SDK1"***, and ***"SDK2"***. The first one is a trivial all-zero (blank) grid – not a valid Sudoku but interesting nonetheless. The second is a simple easy case, which is resolved quickly. The third however will require a much longer execution time – and all the three of them can be used to check that the programs are working fine. Refer to JM's pages for the grid definition and solutions of these examples.

Example1: Solve the following sudoku:

```
0 0 0 | 3 0 0 | 0 5 0
0 0 5 | 4 0 6 | 0 0 2
2 7 0 | 0 1 0 | 3 6 0
--------------------------
7 0 4 | 2 3 0 | 0 0 0
5 1 0 | 0 0 0 | 0 3 7
0 0 0 | 0 4 7 | 9 0 1
--------------------------
0 4 6 | 0 9 0 | 0 1 5
1 0 0 | 6 0 8 | 7 0 0
0 5 0 | 0 0 4 | 0 0 0
```

A more difficult example: With the grid:

```
0 9 0 | 0 4 2 | 0 1 0
0 0 5 | 0 0 0 | 0 0 0
3 0 0 | 0 0 0 | 9 0 4
--------------------------
0 0 0 | 0 0 0 | 1 9 3
5 2 0 | 7 0 0 | 0 0 6
0 0 0 | 0 0 1 | 0 0 0
--------------------------
9 0 0 | 0 5 0 | 0 6 0
0 0 0 | 2 0 4 | 0 0 7
0 0 0 | 0 1 6 | 8 0 0
```

III. Outputting the results.

Data output is presented in the same compact mode way in all cases – regardless of the program used. There is one prompt per row, repeated until the complete grid is shown. This facilitates reading the solution as it avoids multiple prompts to see the elements of a single row. Below are some examples taken from the solution of the "SDK1" puzzle.

The actual register allocation is detailed in the following table. Note that all data handling is managed by the programs, *so it's completely transparent to the user*. However it's important to know when the individual components are used as subroutines in other programs - or manually from the keyboard.

Program	Input Data Location	Output Data Location
^SROW	Alpha	R1-R9
SUD	R1-R9	R10-R18
SDK	R10-R18	R1-R9
SUDVIEW	R1-R9	unchanged
SDKIN	Alpha (calls ^SROW)	R1-R81
SLSDK	R1-R81	R1-R81
SDKOUT	R1-R81	unchanged

As you can see "***SUDOKU"*** features a much more efficient RAM usage, with the 81 cells stored in just nine data registers – a "compact mode" with each element taking up just one nibble of the corresponding mantissas; behold the power of MCODE in action. In contrast, "***SUDRPN"*** requires many more available registers to execute – *even if the data entry is also done in the same fashion.*

Refer to the Sudoku_Blueprint document for programming code details. The following pages show the program listing for the main SUDOKU program, where you can see how all elements come together to deliver *"something bigger than the sum of its parts"* – or at least close enough.

1	SUDOKU	*FOCAL*	A1C3	*016*	*UserCode: 158 bytes*	
2	SUDOKU	*FOCAL*	A1C4	*240*	*(22 regs + 4 bytes) NONPRIVATE*	
3	SUDOKU	*Header*	A1C5	*1CC*	LABEL	
4	SUDOKU	*Header*	A1C6	*000*	GLOBAL	
5	SUDOKU	*Header*	A1C7	*0F7*	<7-Chrs.>	
6	SUDOKU	*Header*	A1C8	*000*	**""**	
7	SUDOKU	*Header*	A1C9	*053*	**"S"**	***Driver for Data Entry***
8	SUDOKU	*Header*	A1CA	*055*	**"U"**	***and puzzle resolution***
9	SUDOKU	*Header*	A1CB	*044*	**"D"**	***plus review (!)***
10	SUDOKU	*Header*	A1CC	*04F*	**"O"**	
11	SUDOKU	*Header*	A1CD	*04B*	**"K"**	*Ángel Martin*
12	SUDOKU	*Header*	A1CE	*055*	**"U"**	
13	SUDOKU	SUDOKU	A1CF	*1A6*	*XROM 25,44*	
14	SUDOKU		A1D0	*06C*	*A6:6C*	*SIZE?*
15	SUDOKU		A1D1	112	*2*	
16	SUDOKU		A1D2	010	*0*	
17	SUDOKU		A1D3	145	X>Y?	
18	SUDOKU		A1D4	*1A6*	*XROM 25,30*	
19	SUDOKU		A1D5	*05E*	*A6:5E*	*PSIZE*
20	SUDOKU		A1D6	119	*9*	*row counter*
21	SUDOKU		A1D7	*1A4*	*XROM 16,05*	
22	SUDOKU		A1D8	*005*	*A4:05*	*E3/E+*
23	SUDOKU		A1D9	*1A6*	*XROM 25,50*	
24	SUDOKU		A1DA	*072*	*A6:72*	*CLRGX*

Next section show the data entry loops, featuring **^SROW** and **ANUMDL** as main components:- To read each cell value into a nibble of the register we use the trusty old **10^X** function, with an element counter increasing from 1 to 9 as exponent. The message *"LOADING…"* is shown during the processing of each row, to provide feedback to the user that there's some action happening.

Note that pressing BackArrow will not cancel the process, but rather will mode the execution to the next row – until the complete grid is covered. If you want to abort the process simple press the ON key to switch the calculator off.

SUDOKU		A1DB	102	LBL 01	
SUDOKU		A1DC	11B	*E*	*reset build to 1,000000000*
SUDOKU		A1DD	191	STO IND Y (2)	*store in proper location*
SUDOKU		A1DE	0F2		
SUDOKU		A1DF	177	CLX	
SUDOKU		A1E0	119	*9*	
SUDOKU		A1E1	*1A4*	*XROM 16,05*	
SUDOKU		A1E2	*005*	*A4:05*	*E3/E+*
SUDOKU		A1E3	*1A4*	*XROM 16,01*	*Input row elements*
SUDOKU		A1E4	*001*	*A4:01*	*^SROW*
SUDOKU		A1E5	1A9	CF 22	
SUDOKU		A1E6	016		
SUDOKU		A1E7	*1A4*	*XROM 16,00*	*"Loading…" msg*
SUDOKU		A1E8	*000*	*A4:00*	*-SUDOKU*
SUDOKU		A1E9	101	LBL 00	
SUDOKU		A1EA	*1A4*	*XROM 16,04*	*get number in X*
SUDOKU		A1EB	*004*	*A4:04*	*ANUMDL*
SUDOKU		A1EC	1AB	FC?C 22	*got something?*
SUDOKU		A1ED	016		
SUDOKU		A1EE	*1B3*	*GTO 02*	*no, skip row*
SUDOKU		A1EF	*090*	*<Distance>*	*16 bytes*
SUDOKU		A1F0	167	X=0?	*is it zero?*
SUDOKU		A1F1	*1B6*	*GTO 05*	*skip processing*
SUDOKU		A1F2	*087*	*<Distance>*	*7 bytes*
SUDOKU		A1F3	190	RCL Y (2)	
SUDOKU		A1F4	072		
SUDOKU		A1F5	168	INT	*store in corresponding*
SUDOKU		A1F6	157	10^X	*decimal digit - order*
SUDOKU		A1F7	143	/	
SUDOKU		A1F8	192	ST+ IND Z (1)	add to build
SUDOKU		A1F9	0F1		
SUDOKU		A1FA	106	LBL 05	
SUDOKU		A1FB	175	RDN	*discard value*
SUDOKU		A1FC	196	ISG X (3)	*increase element count*
SUDOKU		A1FD	073		
SUDOKU		A1FE	*1B1*	*GTO 00*	*go and fetch next element*
SUDOKU		A1FF	*017*	*<Distance>*	*-23 bytes*
SUDOKU		A200	103	LBL 02	
SUDOKU		A201	175	RDN	*discard element counter*
SUDOKU		A202	196	ISG X (3)	*increase row count*
SUDOKU		A203	073		
SUDOKU		A204	*1B2*	*GTO 01*	*go to next row*
SUDOKU		A205	*02B*	*<Distance>*	*-43 bytes*

After this comes the actual resolution of the puzzle, and the data output routine. **SDK** also produces the *"SOLVING…"* message, which will blink with each iteration of the code.

Header	A206	1C4	LABEL	
Header	A207	009	GLOBAL	
Header	A208	0F4	<4-Chrs.>	
Header	A209	000	""	moves data from R1-R9
Header	A20A	053	"S"	to R10-R19, then calls SDK
Header	A20B	055	"U"	
Header	A20C	044	"D"	
SUD	A20D	119	9	
	A20E	1A4	XROM 16,05	1,009
	A20F	005	A4:05	E3/E+
	A210	176	LASTX	
	A211	140	+	10,009
	A212	1A4	XROM 16,05	1,010009
	A213	005	A4:05	E3/E+
	A214	1A6	XROM 25,35	
	A215	063	A6:63	REGMOVE
	A216	1A4	XROM 16,07	
	A217	007	A4:07	SDK
	A218	1A4	XROM 16,23	rasping sound
	A219	017	A4:17	RASP
	A21A	1FB	Text-11	
	A21B	020	" "	
	A21C	02A	"*"	
	A21D	02A	"*"	
	A21E	020	" "	
	A21F	044	"D"	
	A220	04F	"O"	" ** DONE **"
	A221	04E	"N"	
	A222	045	"E"	
	A223	020	" "	
	A224	02A	"*"	
	A225	02A	"*"	
	A226	17E	AVIEW	
	A227	189	PSE	

Only remaining part is the data output. Results are stored in R1-R9, so it's just about prompting them in a similar fashion as the used during the data entry process.

Header	A228	1CC	LABEL	
Header	A229	004	GLOBAL	
Header	A22A	0F8	<8-Chrs.>	
Header	A22B	000	""	
Header	A22C	053	"S"	
Header	A22D	055	"U"	***Views Sudoku***
Header	A22E	044	"D"	stored in R1-R9
Header	A22F	056	"V"	
Header	A230	049	"I"	
Header	A231	045	"E"	*Ángel Martin*
Header	A232	057	"W"	
SUDVIEW	A233	119	9	row counter
	A234	1A4	XROM 16,05	
	A235	005	A4:05	E3/E+
	A236	104	LBL 03	
	A237	1F1	Text-1	
	A238	052	"R"	
	A239	1A4	XROM 16,03	
	A23A	003	A4:03	AINT
	A23B	1F3	Text-3	
	A23C	07F	"|-"	
	A23D	03A	":"	
	A23E	020	" "	
	A23F	119	9	element counter
	A240	1A4	XROM 16,05	
	A241	005	A4:05	E3/E+
	A242	190	RCL IND Y (2)	
	A243	0F2		
	A244	105	LBL 04	
	A245	169	FRC	discard integer part
	A246	11B	E	
	A247	011	1	
	A248	142	*	first decimal to IP
	A249	1A4	XROM 16,03	
	A24A	003	A4:03	AINT
	A24B	1F2	Text-2	append and separate
	A24C	07F	"|-"	
	A24D	03A	":"	
	A24E	196	ISG Y (2)	increase element count
	A24F	072		
	A250	1B5	GTO 04	next element
	A251	00E	<Distance>	-14 bytes
	A252	17E	AVIEW	
	A253	175	RDN	
	A254	175	RDN	
	A255	196	ISG X (3)	increase row count
	A256	073		
	A257	1B4	GTO 03	next row
	A258	023	<Distance>	-35 bytes
	A259	1C0	END	
FOCAL	A25A	007	<CHAIN> -49 bytes	
FOCAL	A25B	22F	<End of Program>	

IV. The 41 strikes back: GRID generation.

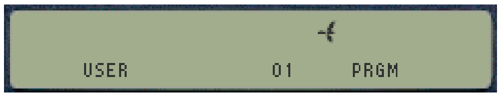
Program **GRID** will prepare a Sudoku grid by randomly clearing some of the elements in a solved Sudoku pre-loaded for this purpose. The number of zeroed elements is defined in X before calling **GRID**. The final grid will be place in compact mode (each element a nibble of the mantissa) in registers R1-R9, ready for ***"SUDOKU"*** in case you have given in.

GRID uses **RNG** to determine which elements within each row will be cleared. RNG uses the TIME Module – so this function will fail if the timer is not present. When the execution ends the message *"GRID MADE"* is shown in the display.

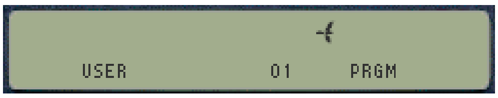
=

Observant users will no doubt note that **GRID** is a left-handed program. The 41 knows that and instructs the goose to behave accordingly – flipping left and running backwards! – all thanks to functions **LEFT** and GOOSE, written by Nelson F. Crowle, one of the authors of the AECROM module among other landmarks.

...



The usage of these functions is shown in the following code snippets. First GOOSE puts up the left goose on the display, and LEFT then should be called at every iteration of a loop, so that the display contents is shifted one position to the left. Note also the other combinations (*) to amuse your friends.



LBL 10	Left Goose flies left
GOOSE	
LBL 01	
LEFT	
0	
GTO 01	
LBL 11	Alpha Rotates Left
AVIEW	
LBL 02	
LEFT	
0	
GTO 02	
LBL 12	Right Goose Flies LEFT
50	
TOGF	
LBL 03	
LEFT	
0	
GTO 03	
LBL 13	Left Goose Flies RIGHT
GOOSE	
50	
TOGF	
LBL 04	
0	
GTO 04	

*(*) Use SF 25, SF 99, and **AVIEW** to rotate Alpha RIGHT.*

TOFG toggles flag status, available in other ROMS.

Outro:- The "Sound F/X" section

To complement the module there is a selection of programs from Jean-Marc's library, providing a nice sample featuring the 41 Beeper in the starring role – capable even if limited as it is. Included is a nice rendition of the Morse code player, plus the Star-Spangled Banner tune and a couple of beep-replacement functions (RASP and CLAXON).

Also related to sound is the Chords-to-Notes conversion programs, a gem for those musicians amongst us with an inclination for unusual chords and scales. A copy of the web pages is included to this mini-manual for completion.

Last and least there is the few utility functions used to round up the programs and save bytes – which by itself makes them worth adding to this and any module. Usual suspects also present in many other modules and probably familiar names by now: **ΣDIG**, **ADEL**, **AINT**, **E3/E+**. A fixture, well worth the space they occupy.

1	ΣDIG	*Header*	A7DB	*087*	*"G"*	
2	ΣDIG	*Header*	A7DC	*009*	*"I"*	
3	ΣDIG	*Header*	A7DD	*004*	*"D"*	
4	ΣDIG	*Header*	A7DE	*04E*	*"Σ"*	*Ángel Martin*
5	ΣDIG	**ΣDIG**	**A7DF**	**0F8**	**READ 3(X)**	
6	ΣDIG		A7E0	00E	A=0 ALL	*initial sum =0*
7	ΣDIG		A7E1	39C	PT= 0	
8	ΣDIG	NEXTD	A7E2	33C	RCR 1	
9	ΣDIG		A7E3	3C6	RSHFC S&X	
10	ΣDIG		A7E4	3C6	RSHFC S&X	
11	ΣDIG		A7E5	146	A=A+C S&X	*add to previous sum*
12	ΣDIG		A7E6	3DC	PT=PT+1	
13	ΣDIG		A7E7	0D4	?PT= 10	
14	ΣDIG		A7E8	*3D3*	*JNC -06*	*[NEXTD]*
15	ΣDIG		A7E9	*17D*	*?NC GO*	***CX-only!!***
16	ΣDIG		A7EA	*0C6*	*->315F*	*[ATOX20]*

1	BUZZER	*Header*	A7EC	*090*	*"P"*	
2	BUZZER	*Header*	A7ED	*013*	*"S"*	
3	BUZZER	*Header*	A7EE	*001*	*"A"*	
4	BUZZER	*Header*	A7EF	*012*	*"R"*	*Mark Power*
5	BUZZER	**RASP**	**A7F0**	**3B8**	**READ 14(d)**	
6	BUZZER		A7F1	17C	RCR 6	
7	BUZZER		A7F2	3D8	C<>ST	
8	BUZZER		A7F3	08C	?FSET 5	
9	BUZZER		A7F4	3A0	?NC RTN	
10	BUZZER		A7F5	130	LDI S&X	
11	BUZZER		A7F6	0FF	CON:	
12	BUZZER		A7F7	358	ST=C	
13	BUZZER		A7F8	2D8	ST<>T	
14	BUZZER		A7F9	130	LDI S&X	
15	BUZZER		A7FA	048	CON:	
16	BUZZER		A7FB	2D8	ST<>T	
17	BUZZER		A7FC	106	A<>C S&X	
18	BUZZER		A7FD	1A6	A=A-1 S&X	
19	BUZZER		A7FE	*3FB*	*JNC -01*	
20	BUZZER		A7FF	266	C=C-1 S&X	
21	BUZZER		A800	*3DB*	*JNC -05*	
22	BUZZER		A801	3C4	ST=0	
23	BUZZER		A802	3B8	READ 14(d)	
24	BUZZER		A803	17C	RCR 6	
25	BUZZER		A804	2D8	ST<>T	
26	BUZZER		A805	3E0	RTN	

hp41programs

Home

Sudoku for the HP-41

Overview

1°) Focal Program
2°) M-Code Routine
3°) Creating a Grid

-A sudoku is a 81-cell grid with 9 rows, 9 columns and 9 regions of 3 x 3 cells.
-Each row, each column and each region must contain all the integers from 1 to 9 exactly once, for instance:

```
4 6 1 | 3 7 2 | 8 5 9
9 3 5 | 4 8 6 | 1 7 2
2 7 8 | 9 1 5 | 3 6 4
---------------------------
7 9 4 | 2 3 1 | 5 8 6
5 1 2 | 8 6 9 | 4 3 7
6 8 3 | 5 4 7 | 9 2 1
--------------------------
8 4 6 | 7 9 3 | 2 1 5
1 2 9 | 6 5 8 | 7 4 3
3 5 7 | 1 2 4 | 6 9 8
```

-Given a partially empty grid, the puzzle consists in finding the missing numbers.
-The sudoku above is the solution of the problem below, where the empty cells are replaced by zeros:

```
0 0 0 | 3 0 0 | 0 5 0
0 0 5 | 4 0 6 | 0 0 2
2 7 0 | 0 1 0 | 3 6 0
---------------------------
7 0 4 | 2 3 0 | 0 0 0
5 1 0 | 0 0 0 | 0 3 7
0 0 0 | 0 4 7 | 9 0 1
--------------------------
0 4 6 | 0 9 0 | 0 1 5
1 0 0 | 6 0 8 | 7 0 0
0 5 0 | 0 0 4 | 0 0 0
```

1°) Focal Program

-"SDK" solves a sudoku by backtracking.
-Though it could theoretically solve any - solvable - sudoku, only a few can be solved, due to prohibitive execution times !

Data Registers:

R00 & R82 to R94: temp *(Registers R01 thru R81 are to be initialized before executing "SDK*

• R01 to • R81 = the elements of the sudoku grid, in column order (or row order) with 0 in the empty cells.

Flags: /

Subroutines: /

-Lines 99-102-119 are 3-byte GTOs

01 LBL "SDK"	26 VIEW 00	51 RCL 82	76 RCL 86	
02 3	27 RCL 88	52 X=Y?	77 *	
03 STO 86	28 STO 82	53 GTO 05	78 STO 85	101 DSE 82
04 6.009	29 LBL 02	54 RCL IND 85	79 RCL 86	102 GTO 02
05 STO 87	30 RCL 00	55 ABS	80 -	103 LBL 06
06 9	31 RCL 94	56 X=Y?	81 RCL 92	104 RCL 00
07 STO 88	32 -	57 GTO 05	82 /	105 RCL 94
08 7	33 STO 85	58 DSE 93	83 ST+ 85	106 +
09 STO 89	34 RCL 88	59 CLX	84 LBL 04	107 STO 00
10 73.00009	35 ST/ 85	60 DSE 85	85 RCL IND 85	108 RCL 91
11 STO 90	36 MOD	61 GTO 03	86 ABS	109 X=Y?
12 82	37 STO 83	62 RCL 86	87 RCL 82	110 SF 41
13 STO 91	38 RCL 90	63 X<> 83	88 X=Y?	111 RCL IND 00
14 STO 00	39 +	64 RCL 86	89 GTO 05	112 X>0?
15 E3	40 X<> 85	65 /	90 DSE 85	113 GTO 06
16 STO 92	41 INT	66 INT	91 GTO 04	114 CHS
17 SIGN	42 STO 84	67 RCL 84	92 RCL 87	115 ST+ IND 00
18 STO 94	43 RCL 94	68 RCL 86	93 ST- 85	116 STO 82
19 LBL 01	44 +	69 /	94 DSE 83	117 VIEW 00
20 DSE 00	45 RCL 88	70 INT	95 GTO 04	118 DSE 82
21 X=0?	46 *	71 RCL 88	96 X<>Y	119 GTO 02
22 RTN	47 STO 93	72 *	97 CHS	120 GTO 06
23 RCL IND 00	48 LBL 03	73 +	98 STO IND 00	121 END
24 X#0?	49 RCL IND 93	74 RCL 89	99 GTO 01	
25 GTO 01	50 ABS	75 +	100 LBL 05	

(214 bytes / SIZE 095)

STACK	INPUTS	OUTPUTS
X	/	/

Example1: Solve the following sudoku:

```
0 0 0 | 3 0 0 | 0 5 0
0 0 5 | 4 0 6 | 0 0 2
2 7 0 | 0 1 0 | 3 6 0
-------------------------
7 0 4 | 2 3 0 | 0 0 0
5 1 0 | 0 0 0 | 0 3 7
0 0 0 | 0 4 7 | 9 0 1
-------------------------
0 4 6 | 0 9 0 | 0 1 5
1 0 0 | 6 0 8 | 7 0 0
0 5 0 | 0 0 4 | 0 0 0
```

-Store the 81 elements into R01 to R81 in column order (0 STO 01 STO 02 2 STO 03 .... 0 STO 81)

XEQ "SDK" >>>> the solution hereunder is returned in about 1h02mn (!)

```
4 6 1 | 3 7 2 | 8 5 9
9 3 5 | 4 8 6 | 1 7 2
```

```
2 7 8 | 9 1 5 | 3 6 4
---------------------------
7 9 4 | 2 3 1 | 5 8 6
5 1 2 | 8 6 9 | 4 3 7
6 8 3 | 5 4 7 | 9 2 1
--------------------------
8 4 6 | 7 9 3 | 2 1 5
1 2 9 | 6 5 8 | 7 4 3
3 5 7 | 1 2 4 | 6 9 8
```

Actually, the digits found by the HP-41 are preceded by a "minus" sign

Example2: With the empty grid - which is not a "proper" sudoku - "SDK" gives the following solution:

XEQ "CLRG"
XEQ "SDK" >>>> (in about 2h30mn)

```
8 9 2 | 5 6 3 | 4 7 1
6 7 3 | 4 9 1 | 5 8 2
4 5 1 | 7 8 2 | 6 9 3
---------------------------
9 2 8 | 6 3 5 | 7 1 4
7 3 6 | 9 1 4 | 8 2 5
5 1 4 | 8 2 7 | 9 3 6
--------------------------
2 8 9 | 3 5 6 | 1 4 7
3 6 7 | 1 4 9 | 2 5 8
1 4 5 | 2 7 8 | 3 6 9
```

Notes:

-Obviously, this program is very slow, all the more that these examples belong to the easy puzzles.
-However, if you are using a good emulator in turbo mode, the execution times become about 6 seconds and 15 seconds respecti

-The address of the current register is displayed (81 80 ....)
-If line 110 is executed, displaying "NONEXISTENT", the puzzle has no solution.

2°) M-Code Routine

-This M-Code routine uses a similar scheme to solve a sudoku.
-It is almost 200 times as fast as the focal program.
-So, much more puzzles can be solved in a "raisonnable" time.

```
08B  "K"          @E347  in my ROM
004  "D"
013  "S"
3C8  CLRKEY
378  C=c
03C  RCR 3
106  A=C S&X
130  LDI S&X
012  012h=018d
146  A=A+C S&X
130  LDI S&X
200  200h=512d
306  ?A<C S&X
381  ?NCGO
00A  NONEXISTENT
378  C=c
0A6  A<>C S&X
106  A=C S&X
1BC  RCR 11
130  LDI S&X
```

```
009   009
246   C=A-C S&X
106   A=C S&X
27C  RCR 9
0A6  A<>C S&X
0BC  RCR 5
070   N=C ALL          here N contains  the addresses of  R09  R00  R18  R09  in nybbles   11-10-9  8-7-6  5-4-3  2-1-0  respecti
04E   C=0 ALL
19C   PT=11
050   LD@PT- 1
050   LD@PT- 1
050   LD@PT- 1
050   LD@PT- 1
050   LD@PT- 1         C = 111111111   in nybbles  11 to 3
050   LD@PT- 1
050   LD@PT- 1
050   LD@PT- 1
050   LD@PT- 1
268   Q=C
10E   A=C ALL
1EE   C=C+C  ALL
1EE   C=C+C  ALL
1EE   C=C+C   ALL
20E   C=A+C ALL
228   P=C              P = 999999999   in nybbles  11 to 3
0A0  SLCT P
21C  PT=2              loop 0
3DC  PT=PT+1           LOOP1 at the address  E376  in my ROM
0B0   C= N ALL
354   ?PT=12
063   JNC+12d
266   C=C-1 S&X
27A  C=C-1 M
070   N=C ALL
106   A=C S&X
17C  RCR 6
366   ?A#C S&X
3AF   JC-11d           goto loop 0
04E   C=0 ALL
270   RAMSLCT
228   P=C
3E0   RTN              the routine stops here if a solution is found
270   RAMSLCT
038   READATA
2E2   ?C#0@PT
377   JC-18d           goto loop 1
10E   A=C ALL
046   C=0 S&X
270   RAMSLCT
238   C=P
158   M=C ALL          LOOP2  at the address  E38D in my ROM
0E0   SLCT Q
01C   PT=3
362   ?A#C@PT              we test if the digit is different from all the other digits in the same column
07B   JNC+15d
3DC  PT=PT+1
354   ?PT=12
3E3   JNC-04
0A0   SLCT P
130   LDI S&X
008   008
10E   A=C ALL
0B0   C=N ALL
17C   RCR 6
226    C=C+1 S&X
270   RAMSLCT
0E6   B<>C S&X
```

```
038  READATA
362  ?A#C@PT        we test if the digit is different from all the other digits in the same row
1A3  JNC+52d
0E6  B<>C S&X
1A6  A=A-1 S&X
3C3  JNC-08
0B0  C=N ALL
10E  A=C ALL
1A6  A=A-1 S&X
17C  RCR 6
226  C=C+1 S&X
226  C=C+1 S&X
226  C=C+1 S&X
306  ?A<C S&X
03F  JC+07
226  C=C+1 S&X
226  C=C+1 S&X
226  C=C+1 S&X
306  ?A<C S&X
017  JC+02
03C  RCR 3
0E6  B<>C S&X       here, B S&X contains the address of the register in the right lower corner of the 3x3 region ( R09 or R
0E0  SLCT Q
01C  PT=3
0A0  SLCT P
014  ?PT=3
087  JC+16d
054  ?PT=4
077  JC+14d
094  ?PT=5
067  JC+12d
0E0  SLCT Q
15C  PT=6
0A0  SLCT P
154  ?PT=6
03F  JC+07
294  ?PT=7
02F  JC+05
114  ?PT=8
01F  JC+03
0E0  SLCT Q
25C  PT=9
0E0  SLCT Q         now, pointer Q points to the right lower corner of the 3x3 region ( 9 or 6 or 3 )
198  C=M
130  LDI S&X        the 4 instructions on the left prepare 2 loops that may be executed 3 times
002  002
23E  C=C+1 MS
23E  C=C+1 MS
10E  A=C ALL
0E6  B<>C S&X
270  RAMSLCT
0E6  B<>C S&X
038  READATA
362  ?A#C@PT        we test if the digit is different from all the other digits in the same 3x3 region
0BB  JNC+23d
3DC  PT=PT+1
1A6  A=A-1 S&X
3E3  JNC-04
166  A=A+1 S&X
3D4  PT=PT-1
166  A=A+1 S&X
3D4  PT=PT-1
166  A=A+1 S&X
3D4  PT=PT-1
0E6  B<>C S&X
266  C=C-1 S&X
0E6  B<>C S&X
```

```
1BE  A=A-1 MS
36B  JNC-19
0A0  SELCT P
0B0  C=N ALL         if the candidate number has successfully passed all the tests, it replaces the zero in the empty cell
270  RAMSLCT
038  READATA
0A2  A<>C@PT
2F0  WRITDATA
1D9  ?NCGO           Change these words written in red according to the address of LOOP1 in your own ROM
38E  LOOP1
0A0  SLCT P          the execution jumps here if the digit has been rejected
198  C=M ALL
10E  A=C ALL
046  C=0 S&X
270  RAMSLCT
278  C=Q
1CE  A=A-C ALL       we started with 999999999 in M, then we try 888888888 in M , ...etc...
0B0  C=N ALL
270  RAMSLCT
038  READATA
0AE  A<>C ALL
2EE  ?C#0 ALL        ... until we arrive at  000000000
235  ?CGO            Change these words written in red according to the address of LOOP2 in your own ROM
38F  LOOP2
3D4  PT=PT-1         if all the digits are rejected for this cell, we go to the previous cell ( backtracking )
214  ?PT=12
10F  JC+33d          we must also move to the previous register if we were at the left of a register
0B0  C=N ALL
03C  RCR 3
270  RAMSLCT         otherwise, we check in a register between R10 and R18 if the cell was empty or not
038  READATA
2E2  ?C#0@PT
3C7  JC-08           If the cell was not empty, we go to the previous cell again
0B0  C=N ALL
270  RAMSLCT
038  READATA
10E  A=C ALL
04E  C=0 ALL
0A2  A<>C@PT
0AE  A<>C ALL
2F0  WRITDATA        the number in the non-fixed cell is replaced by 0
1A2  A=A-1@PT
342  ?A#0@PT
36B  JNC-19d         if the last tested digit was 1, we again go to the previous non-fixed cell
046  C=0 S&X
270  RAMSLCT
278  C=Q
08E  B=A ALL
10E  A=C ALL
322  ?A<B@PT
01B  JNC+03          if the digit in the previous cell was, say 8, we must recreate 777777777 which will be stored in CPU regis
14E  A=A+C ALL
3EB  JNC-03
0B0  C=N ALL
270  RAMSLCT
038  READATA
0AE  A<>C ALL
235  ?NCGO           Change these words written in red according to the address of LOOP2 in your own ROM
38E  LOOP2
35C  PT=12           we arrive here if we must go to the "previous" register ( R01->R02->R03 ... )
0B0  C=N ALL
226  C=C+1 S&X
23A  C=C+1 M
070  N=C ALL
27C  RCR 9
106  A=C S&X
0BC  RCR 5
```

```
160  ?LOWBAT              the 4 instructions written in yellow on the left
360  ?C RTN               will stop the routine if the batteries are low
3CC  ?KEY                 or if you press any key
360  ?CRTN                They may be deleted if you have a "newest" HP-41:  simply press  ENTER^  ON   to stop the program
306  ?A<C S&X
283  JNC-48d              Replace this line by    2A3  JNC-44d    if you don't key in the yellow instructions
0B5  ?NCGO
0A2  DATA ERROR           if the "previous" register is R10, the sudoku cannot be solved !      ( @E42B in my ROM )
```

(229 words / SIZE 019)

STACK	INPUTS	OUTPUTS
X	/	/

-To use the M-Code routine, we must place the cells in nybbles 11 to 3 of registers R01 to R09 and the same data in R10 to R18.
-So the cells of a row must be the fractional part of a real number whose integer part is between 1 and 9

-The short routine hereunder lets the HP-41 deal with the registers R10 to R18.

```
01  LBL "SUD"
02  1.010009
03  REGMOVE
04  SDK
05  END
```

Examples: The examples of the 1st paragraph are now solved in 22 seconds and 50 seconds respectively.

A more difficult example: With the grid:

```
0 9 0 | 0 4 2 | 0 1 0
0 0 5 | 0 0 0 | 0 0 0
3 0 0 | 0 0 0 | 9 0 4
---------------------------
0 0 0 | 0 0 0 | 1 9 3
5 2 0 | 7 0 0 | 0 0 6
0 0 0 | 0 0 1 | 0 0 0
--------------------------
9 0 0 | 0 5 0 | 0 6 0
0 0 0 | 2 0 4 | 0 0 7
0 0 0 | 0 1 6 | 8 0 0
```

```
1.090042010   STO 01
1.005000000   STO 02
1.300000904   STO 03
1.000000193   STO 04
1.520700006   STO 05
1.000001000   STO 06
1.900050060   STO 07
1.000204007   STO 08
1.000016800   STO 09    XEQ "SUD"  ( not SDK )  returns a solution in 42mn44s
```

-The same solution is returned in 1mn28s (!) if we store the columns instead of the rows.

(1.003050900 STO 01 1.900020000 STO 02 .... 1.004360070 STO 09)

-The solution is in registers R01 thru R09:

```
7 9 8 | 6 4 2 | 3 1 5
4 1 5 | 9 7 3 | 6 2 8
3 6 2 | 1 8 5 | 9 7 4
```

```
---------------------------
6 7 4 | 5 2 8 | 1 9 3
5 2 1 | 7 3 9 | 4 8 6
8 3 9 | 4 6 1 | 7 5 2
--------------------------
9 4 3 | 8 5 7 | 2 6 1
1 8 6 | 2 9 4 | 5 3 7
2 5 7 | 3 1 6 | 8 4 9
```

<u>Notes:</u>

-R00 is unused. Synthetic registers P & Q are used. Register P is cleared at the end if a solution has been found.

-Of course, here again, a good emulator like V41 in turbo mode will reduce the execution times by a factor about 600.
-So, many more puzzles can be solved in this case.
-Several minutes may still be required however (perhaps hours in very difficult cases ...)

-You can also help your HP-41 in many ways:
-For example, if the last row is empty and the 8th row is not, swap them (R08 <> R09)
 and swap these registers again when the solution is found.
-Remember that the search starts with the last digit of register R09 then the next to last (i-e 8th) digit of R09 and so on ...

3°) Creating a Grid

-This program uses a solved sudoku (lines 08 to 25), then it randomly permutes raws and columns
 and stores this - solved - sudoku in registers R19 to R27 (lines 127-128).
-Finally, cells are replaced by 0 to get a puzzle with N fixed cells, where N is to be specified by the user.

-Line 126 is a three-byte GTO 01.

01 LBL "GRID"	35 R-D	69 LBL 05	103 RCL 11	137 9
02 STO 00	36 FRC	70 RCL 00	104 /	138 *
03 81	37 STO 00	71 R-D	105 STO 17	139 INT
04 RCL Z	38 3	72 FRC	106 *	140 1
05 -	39 *	73 STO 00	107 STO Y	141 +
06 INT	40 INT	74 3	108 10	142 10^X
07 STO 18	41 LBL 03	75 *	109 MOD	143 STO 12
08 1.798642315	42 RCL 00	76 INT	110 INT	144 RCL IND 11
09 STO 01	43 R-D	77 X=Y?	111 ST- Y	145 1
10 1.415973628	44 FRC	78 GTO 05	112 X<> 13	146 X=Y?
11 STO 02	45 STO 00	79 RCL 10	113 +	147 GTO 09
12 1.362185974	46 3	80 +	114 RCL 17	148 RDN
13 STO 03	47 *	81 INT	115 /	149 *
14 1.674528193	48 INT	82 10^X	116 RCL 13	150 STO Y
15 STO 04	49 X=Y?	83 STO 11	117 +	151 10
16 1.521739486	50 GTO 03	84 CLX	118 RCL 11	152 MOD
17 STO 05	51 RCL 10	85 RCL 10	119 /	153 INT
18 1.839461752	52 ST+ Z	86 +	120 STO IND 14	154 X=0?
19 STO 06	53 +	87 INT	121 DSE 14	155 GTO 08
20 1.943857261	54 RCL IND Y	88 10^X	122 GTO 06	156 -
21 STO 07	55 X<> IND Y	89 STO 12	123 DSE 10	157 RCL 12
22 1.186294537	56 STO IND Z	90 9	124 GTO 04	158 /
23 STO 08	57 DSE 10	91 STO 14	125 DSE 16	159 STO IND 11
24 1.257316849	58 GTO 02	92 LBL 06	126 GTO 01	160 DSE 18
25 STO 09	59 RCL 15	93 RCL IND 14	127 1.019009	161 X=0?
26 7.00003	60 STO 10	94 RCL 11	128 REGMOVE	162 GTO 10
27 STO 15	61 LBL 04	95 *	129 LBL 07	163 LBL 09
28 5	62 RCL 00	96 STO Y	130 9	164 DSE 11
29 STO 16	63 R-D	97 10	131 STO 11	165 GTO 08
30 LBL 01	64 FRC	98 MOD	132 LBL 08	166 GTO 07
31 RCL 15	65 STO 00	99 INT	133 RCL 00	167 LBL 10
32 STO 10	66 3	100 STO 13	134 R-D	168 END

33 LBL 02 34 RCL 00	67 * 68 INT	101 - 102 RCL 12	135 FRC 136 STO 00	

(321 bytes / SIZE 028)

STACK	INPUTS	OUTPUTS
Y	N	/
X	r	/

where N is an integer between 1 and 81 to get a puzzle with N non-empty cells
and r is a random seed

Example: You want to get a sudoku with 28 non-empty cells, and you choose 1 as random seed.

28 ENTER^
1 XEQ "GRID" and we get the grid in registers R01 thru R09

R01 = 1.800600130
R02 = 1.003050004
R03 = 1.000900068
R04 = 1.008016000
R05 = 1.005003800 (the integer part doesn't really matter)
R06 = 1.006500003
R07 = 1.000361000
R08 = 1.600000307
R09 = 1.000005601

-So, the puzzle is

```
8 0 0 | 6 0 0 | 1 3 0
0 0 3 | 0 5 0 | 0 0 4
0 0 0 | 9 0 0 | 0 6 8
---------------------------
0 0 8 | 0 1 6 | 0 0 0
0 0 5 | 0 0 3 | 8 0 0
0 0 6 | 5 0 0 | 0 0 3
--------------------------
0 0 0 | 3 6 1 | 0 0 0
6 0 0 | 0 0 0 | 3 0 7
0 0 0 | 0 0 5 | 6 0 1
```

-If you don't solve the grid, one solution is in registers R19 thru R27
-In this example, "SDK" gives another solution.

Notes:

-Replace line 28 (5) by a larger integer if you think that the original grid is not enough shuffled.
-This routine does not always return a proper sudoku (i-e with a unique solution), especially if N is small.
-If it happens ... it's only by chance !

-If you have a "RAND" M-code function, replace all the RCL 00 R-D FRC STO 00 by RAND
replace lines 02-03-04 by 81 X<>Y
and simply put N in X-register before executing "GRID"

hp41programs

Home

564 Chords for the HP-41

Overview

-The following program (actually 2 routines) performs the Chords <> Notes conversion:
-"C-N" displays the notes of a given chord.
-"N-C" searches the chord(s) corresponding to (3 to 6) given notes *without regard of order.*

-"N-C" identifies 47 types of chords, namely (for example in C):

CMaj , CMin , C- , C/4 , C5+ , C5+ 9 , C5+ 9- , C5- , C6 , CMin6 , C7 , C7- , C7 5+ , C7 5- , CMin7 , CMin7+ , CMin7 5- , CMaj7 , CMaj7 5+
CMaj7 5- , C7/4 , CMin7/4 , C7/6 , CMin7/6 , CMaj7/6 , Cadd9 , CMinadd9 , C9 , CMin9 , C9 5+ , C9 5- , CMaj9 , CMin9 7+ , C9+ , C9- , C9- 5+
C9/6 , CMin9/6 , C11 , CMin11 , C11+ , CMaj11 , C13 , CMin13 , CMaj13 , C13 9- , C13 5-9-

-Other notations are sometimes used, for instance, C13 (b5 b9) = C13 5-9- = thirteenth chord with diminished fifth and diminished ninth ...

-These chords are coded as follows: we use the relations A = 0 , Bb = 1 , B = 2 , ... , Ab = 11 , A = 12 , ...
Let's take for example A11 (eleventh chord)

A11 = A Db E G B D = 0 4 7 10 14 17 whence, modulo 12: 0 4 7 10 2 5 rearranged in increasing order: 0 2 4 5 7 10
then we take the difference between 2 consecutive integers: 2-0 = 2 , 4-2 = 2 , 5-4 = 1 , 7-5 = 2 , 10-7 = 3
which finally yields 22123 for eleventh chords (the dominant doesn't change the final result)
22123 is the content of X-register at line 467 and similarly for the other chords.

-"C-N" finds the same chords, plus a few extra ones ...
"C-N" will work if the left part of the alpha string (after the dominant and a possible MAJ or MIN) is one of the following symbols:
13 11 11+ 9 9+ 9- 7 7+ 7- - /4 5+ 5- and if the rest of the string only contains one (or several) of the characters:
6 4 5+ 5- 7+ 7- 9 9+ 9- 11+

Program Listing

Data Registers:
C-N: R00 = dominant ; R01 = 11th or 13th ; R02 = 9th ; R03 = 7th ; R04 = 6th ; R05 = 5th ; R06 = 4th ; R07 = 3rd ; R08 = 1
N-C: R00 = -------- ; R01 to Rnn = the n notes of the chord (n < 7)
R09-R10-R11: temp

Flags: /
Subroutines: /

-Line 01 = **LBL "C-N"** **Chord >>>> Notes**
-Line 177 = **LBL "N-C"** **Notes >>>> Chord(s)**

-Lines 17 to 21 eliminate the possible spaces (ASCII code = 32) at the left of the string.
-The characters different from A B C D E F G # b are simply ignored.
-The "append" character is denoted ~
-Lines 255 to 271 sort the content of registers R01 thru Rnn in increasing order (n = the number of notes)

-Line 310 may be replaced by " " (1 space)
-Line 361 may be replaced by "7+ 5-"
-Line 373 may be replaced by "7+"
-Line 377 may be replaced by "7+ 5+"
-Line 392 may be replaced by "7 5+9-"
-Line 420 may be replaced by "9 7+"
-Line 445 may be replaced by "7+/6"
-Line 465 may be replaced by "11 7+"
-Line 476 may be replaced by "13 7+"

-Line 502 is a three-byte GTO

```
01 LBL "C-N"
02 CLX
03 STO 04
04 STO 06
05 SIGN
06 STO 08
07 11
08 STO 03
09 ST+ X
10 STO 01
11 15
12 STO 02
13 8
14 STO 05
15 5
16 STO 07
17 LBL 00
18 32
19 ATOX
20 X=Y?
21 GTO 00
22 XEQ 05
23 STO 00
24 CLX
25 LBL 01
26 X#0?
27 SIGN
28 ST+ 00
29 29
30 ATOX
31 X=Y?
32 GTO 01
33 98
34 X=Y?
35 CHS
36 X<0?
37 GTO 01
38 X<>Y
39 XTOA
40 SIGN
41 CHS
42 AROT
43 ASTO 09
44 ASHF
45 ASTO 10
46 "MIN"
47 XEQ 08
48 X#0?
49 GTO 02
50 3
51 AROT
52 ASTO 09
53 ASHF
54 ASTO 10
55 DSE 07
56 LBL 02
57 "MAJ"
58 XEQ 08
59 X#0?
60 GTO 02
61 3
62 AROT
63 ASTO 09
64 ASHF
65 ASTO 10
66 ISG 03
67 LBL 02
68 "13"
69 XEQ 08
70 X=0?
71 GTO 02
72 18
73 STO 01
74 "11"
75 XEQ 08
76 X=0?
77 GTO 02
78 CLX
79 STO 01
85 STO 02
86 55
87 POSA
88 0
89 X#Y?
90 STO 03
91 LBL 02
92 4
93 "6"
94 10
95 XEQ 07
96 "7-"
97 XEQ 08
98 45
99 POSA
100 *
101 X=0?
102 DSE 05
103 X=0?
104 DSE 07
105 52
106 POSA
107 X<0?
108 GTO 02
109 6
110 STO 06
111 LBL 02
112 5
113 "5+"
114 9
115 XEQ 07
116 5
117 "5-"
118 7
119 XEQ 07
120 3
121 "7+"
122 12
123 XEQ 07
124 3
125 "7-"
126 10
127 XEQ 07
128 57
129 POSA
130 X<0?
131 GTO 02
132 15
133 STO 02
134 LBL 02
135 2
136 "9+"
137 16
138 XEQ 07
139 2
140 "9-"
141 14
142 XEQ 07
143 1
144 "11+"
145 19
146 XEQ 07
147 RCL 06
148 RCL 07
149 -
150 1
151 -
152 X=0?
153 STO 07
154 "7/6"
155 XEQ 08
156 X=0?
157 GTO 02
158 RCL 03
159 RCL 04
160 -
161 1
162 -
163 X=0?
169 LBL 03
170 RCL IND 09
171 X#0?
172 XEQ 06
173 DSE 09
174 GTO 03
175 TONE 9
176 PROMPT
177 LBL "N-C"
178 CLX
179 STO 07
180 LBL 04
181 ATOX
182 X=0?
183 GTO 02
184 98
185 X=Y?
186 DSE IND 07
187 CLX
188 29
189 X=Y?
190 ISG IND 07
191 ENTER^
192 68
193 RCL Z
194 -
195 X^2
196 9
197 X<Y?
198 GTO 04
199 R^
200 XEQ 05
201 ISG 07
202 CLX
203 STO IND 07
204 GTO 04
205 LBL 05
206 .61
207 /
208 106
209 -
210 INT
211 RTN
212 LBL 06
213 RCL 00
214 +
215 12
216 MOD
217 STO 11
218 .59
219 *
220 65.1
221 +
222 XTOA
223 RCL 11
224 X=0?
225 "~b"
226 6
227 -
228 ABS
229 4
230 X=Y?
231 "~b"
232 SIGN
233 X=Y?
234 "~b"
235 RTN
236 LBL 07
237 XEQ 08
238 X<0?
239 RTN
240 X<>Y
241 STO IND Z
242 RTN
243 LBL 08
244 ENTER^
245 ASTOX
246 CLA
247 ARCL 09
253 STO 00
254 STO 08
255 LBL 09
256 RCL 00
257 RCL 00
258 RCL IND X
259 LBL 10
260 RCL IND Z
261 X>Y?
262 STO Z
263 X>Y?
264 +
265 RDN
266 DSE Z
267 GTO 10
268 X<> IND 00
269 STO IND Y
270 DSE 00
271 GTO 09
272 LBL 11
273 RCL 07
274 RCL 01
275 ST+ 00
276 LBL 12
277 ST- IND Y
278 DSE Y
279 GTO 12
280 RCL 07
281 .1
282 %
283 2
284 +
285 0
286 STO 09
287 LBL 13
288 10
289 *
290 RCL IND Y
291 ST+ Y
292 X<> 09
293 -
294 ISG Y
295 GTO 13
296 33
297 X=Y?
298 "-"
299 1
300 +
301 X=Y?
302 "MIN"
303 8
304 +
305 X=Y?
306 "5-"
307 1
308 +
309 X=Y?
310 "MAJ"
311 LASTX
312 +
313 X=Y?
314 "5+"
315 8
316 +
317 X=Y?
318 "/4"
319 82
320 +
321 X=Y?
322 "5+ 9-"
323 90
324 STO 09
325 +
326 X=Y?
327 "5+ 9"
328 DSE X
329 X=Y?
330 "add9"
331 9
337 X=Y?
338 "MIN7 5-"
339 DSE X
340 X=Y?
341 "7-"
342 9
343 +
344 X=Y?
345 "MIN6"
346 1
347 +
348 X=Y?
349 "MIN7"
350 LASTX
351 +
352 X=Y?
353 "MIN7+"
354 80
355 +
356 X=Y?
357 "7 5-"
358 1
359 +
360 X=Y?
361 "MAJ7 5-"
362 7
363 +
364 X=Y?
365 "6"
366 1
367 +
368 X=Y?
369 "7"
370 LASTX
371 +
372 X=Y?
373 "MAJ7"
374 9
375 +
376 X=Y?
377 "MAJ7 5+"
378 DSE X
379 X=Y?
380 "7 5+"
381 81
382 +
383 X=Y?
384 "7/4"
385 810
386 +
387 X=Y?
388 "9-"
389 9
390 +
391 X=Y?
392 "9- 5+"
393 800
394 +
395 X=Y?
396 "MIN9/6"
397 1
398 +
399 X=Y?
400 "MIN9"
401 LASTX
402 +
403 X=Y?
404 "MIN9 7+"
405 80
406 +
407 X=Y?
408 "9 5-"
409 8
410 +
411 X=Y?
412 "9/6"
413 1
414 +
415 X=Y?
421 8
422 +
423 X=Y?
424 "9 5+"
425 891
426 +
427 X=Y?
428 "9+"
429 RCL 09
430 +
431 X=Y?
432 "MIN7/4"
433 198
434 STO 10
435 +
436 X=Y?
437 "MIN7/6"
438 900
439 +
440 X=Y?
441 "7/6"
442 1
443 +
444 X=Y?
445 "MAJ7/6"
446 8909
447 +
448 X=Y?
449 "13 5-9-"
450 RCL 09
451 +
452 X=Y?
453 "13 9-"
454 7902
455 +
456 X=Y?
457 "MIN11"
458 RCL 10
459 +
460 X=Y?
461 "MIN13"
462 703
463 +
464 X=Y?
465 "MAJ11"
466 DSE X
467 X=Y?
468 "11"
469 RCL 09
470 +
471 X=Y?
472 "11+"
473 109
474 +
475 X=Y?
476 "MAJ13"
477 DSE X
478 X=Y?
479 "13"
480 ALENG
481 X=0?
482 GTO 02
483 ASTO 09
484 ASHF
485 ASTO 10
486 " "
487 1
488 XEQ 06
489 ARCL 09
490 ARCL 10
491 TONE 9
492 PROMPT
493 LBL 02
494 CLA
495 RCL 07
496 12
497 LBL 14
498 X<> IND Y
499 DSE Y
```

80 57	164 STO 03	248 ARCL 10	332 -	416 "9"	500 GTO 14
81 POSA	165 LBL 02	249 POSA	333 X=Y?	417 LASTX	501 DSE 08
82 X=0?	166 " "	250 RTN	334 "MINadd9"	418 +	502 GTO 11
83 GTO 02	167 8	251 LBL 02	335 CLX	419 X=Y?	503 BEEP
84 CLX	168 STO 09	252 RCL 07	336 334	420 "MAJ9"	504 END

(934 bytes / SIZE 012)

STACK	INPUTS	OUTPUTS
X	/	/

All the inouts/outputs are in the alpha "register"

"C-N" examples: Execution time = 20 to 50 seconds

"CMAJ" (or simply "C")	XEQ "C-N" >>>>	(TONE 9)	" CEG"	3 notes: C E G
"C-"	XEQ "C-N" >>>>	(TONE 9)	" CEbGb"	3 notes: C Eb Gb
"D7-"	XEQ "C-N" >>>>	(TONE 9)	" DFAbB"	4 notes: D F Ab B
"Bb5+ 9-"	XEQ "C-N" >>>>	(TONE 9)	" BbDGbB"	4 notes: Bb D Gb B
"Aadd9" (or "A 9")	XEQ "C-N" >>>>	(TONE 9)	" ADbEB"	4 notes: A Db E B
"A9"	XEQ "C-N" >>>>	(TONE 9)	" ADbEGB"	5 notes: A Db E G B
"C#13 5-9-" (or "Db13 5-9-")	XEQ "C-N" >>>>	(TONE 9)	" DbFGBDBb"	6 notes: Db F G B D Bb

-You can add space(s) at the left or the right of the alpha string or between 2 groups of symbols,
for instance, you can key in: " CMAJ7 5+ " or " CMAJ75+" , but *not* "C MAJ 7 5 +"
and "Bb5+ 9-" but *neither* "Bb 5 + 9-" *nor* "B b 5+9-"

"N-C" examples: Execution time = 17 to 80 seconds

"CEG"	XEQ "N-C"	>>>>	(TONE 9)	" CMAJ"	
	R/S	>>>>	(BEEP)	0	(no other chord)
"DFAbB"	XEQ "N-C"	>>>>	(TONE 9)	" Ab7-"	
	R/S	>>>>	(TONE 9)	" B7-"	
	R/S	>>>>	(TONE 9)	" D7-"	
	R/S	>>>>	(TONE 9)	" F7-"	
	R/S	>>>>	(BEEP)	0	(no other chord)
"ABCDbG#Ab"	XEQ "N-C"	>>>>	(BEEP)	0	(unknown chord)
"A#EbbF#B"	XEQ "N-C"	>>>>	(TONE 9)	" Bb5+ 9-"	
	R/S	>>>>	(TONE 9)	" BMIN7+"	
	R/S	>>>>	(BEEP)	0	(no other chord)

-Strictly speaking, "Bb5+ 9-" = Bb D Gb B and "BMIN7+" = B D Gb Bb are not quite identical.
-Practically, however, they are usually obtained by the same fingerings on a guitar.

Notes: (I mean "remarks")

-Don't key in more than 6 notes before executing "N-C". All these notes must be different.
-You can key in several b and several # (for instance Bbb instead of A , C## instead of D)
-The character "#" (alpha shift SIN) doesn't appear very clearly on the HP-41 (35 XTOA would be better but not very easy to handle)
if you want to replace it by another one, replace lines 29 and 188 by the corresponding ASCII code (for example 100 or E2 if you replace "#" by "d")
-If you use the notations "7+" , "9 7+" , "11 7+" , "13 7+" which are equivalent to
"MAJ7" , "MAJ9" , "MAJ11" , "MAJ13" respectively, lines 57 to 67 may be deleted.
-Several bytes can be saved if you omit some spaces (for example line 449 "135-9-" instead of "13 5-9-") but the display is less legible.

-If you wish that "N-C" identifies another chord, say MIN/4 , insert CLX 322 X=Y? "MIN/4" after line 479
(the intervals between 2 consecutive notes of this chord are 3 2 2)
-Similarly, if you want to delete one of the chords, say MAJ7+ 5- , delete line 358 to 361 *after replacing line 362 by 8* (instead of 7)
so that the next content of X-register is unchanged.

-If you key in "Cadd11" XEQ "C-N" you'll get " CEG" (the "11" will not be taken into account)
-In order to identify this chord, add 1 "11" 18 XEQ 07 after line 146.

hp41programs

Home

Morse Code for the HP-41

Overview

-Clifford Stern has written a superb Morse Code program which is listed in "Synthetic Programming made easy" by Keith Jarett.
-The following "MC" is far from being so good but it uses the ATOX function of the X-Functions module,
it can transmit more characters and it occupies less program memory.
-"MC" uses the synthetic TONE P (decimal codes = 159 , 120) for . and the standard TONE 8 for _
-For instance, L = TONE P TONE 8 TONE P TONE P = . _ . .

-Then, a second routine ("LMC") may help you to learn Morse code.

Warning:

-If the last executed tone is a synthetic tone (TONE P or another one),
my HP-41 emits a strange frrrrrrr (press your ear against your calculator to check)
-Simply execute a BEEP or any non-synthetic TONE to remove this vibration

1°) Morse Code Program

Codes: *(in the same order as the LBLs)*

L = . _ . .	/ = _ . . _ .	: = _ _ _ . . .	. = . _ . _ . _	4 = ... _	
? = . . _ _ . .	F = . . _ .	7 = _ _ . . .	_ = . . _ _ . _	V = . . . _	
8 = _ _ _ . .	R = . _ .	B = _ . . .	) = _ . _ _ . _	U = . . _	1 = . _ _ _ _
Z = _ _ . .	(= _ . _ _ .	6 = _	Q = _ _ . _	A = . _	0 = _ _ _ _ _
D = _ . .	P = . _ _ .	& = . _ . . .	K = _ . _	, = _ _ . . _ _	O = _ _ _
; = _ . _ . _ .	! = . . . _ .	5 = ...	= = _ . . . _	3 = . . . _ _	M = _ _
+ = . _ . _ .	' = . _ _ _ _ .	H = ...	$ = . . . _ . . _	2 = . . _ _ _	T = _
@ = . _ _ . _ .	9 = _ _ _ _ .	S = . . .	X = * = _ . . _	J = . _ _ _	space = a pause
C = _ . _ .	G = _ _ .	I = . .	% = _ . _ . _	Y = _ . _ _	
" = . _ . . _ .	N = _ .	E = .	- = _ . . . . _	W = . _ _	

Data Registers: /
Flags: /
Subroutines: /

01 LBL "MC"	26 LBL 43	51 LBL 39	76 TONE P	101 XEQ 19	126 LBL 50
02 LBL 00	27 XEQ 18	52 TONE P	77 LBL 19	102 LBL 24	127 TONE P
03 64	28 GTO 14	53 LBL 57	78 TONE P	103 LBL 42	128 LBL 10
04 ATOX	29 LBL 64	54 XEQ 13	79 LBL 09	104 TONE 8	129 TONE P
05 X=0?	30 XEQ 01	55 LBL 07	80 TONE P	105 GTO 21	130 GTO 15
06 GTO 32	31 LBL 03	56 TONE 8	81 LBL 05	106 LBL 37	131 LBL 25

07 X>Y?	32 TONE 8	57 LBL 14	82 TONE P	107 XEQ 11	132 TONE 8
08 -	33 GTO 18	58 TONE 8	83 RTN	108 GTO 01	133 LBL 23
09 XEQ IND X	34 LBL 34	59 GTO 05	84 LBL 46	109 LBL 45	134 TONE P
10 GTO 00	35 TONE P	60 LBL 58	85 XEQ 18	110 TONE 8	135 GTO 13
11 LBL 12	36 LBL 47	61 TONE 8	86 GTO 11	111 LBL 52	136 LBL 49
12 TONE P	37 TONE 8	62 LBL 55	87 LBL 31	112 TONE P	137 TONE P
13 GTO 04	38 LBL 06	63 TONE 8	88 XEQ 21	113 LBL 22	138 GTO 02
14 LBL 63	39 TONE P	64 LBL 02	89 GTO 11	114 TONE P	139 LBL 48
15 XEQ 21	40 LBL 18	65 TONE 8	90 LBL 41	115 LBL 21	140 TONE 8
16 GTO 04	41 TONE P	66 GTO 19	91 XEQ 14	116 TONE P	141 LBL 02
17 LBL 56	42 GTO 14	67 LBL 54	92 LBL 17	117 LBL 01	142 TONE 8
18 TONE 8	43 LBL 40	68 TONE 8	93 TONE 8	118 TONE P	143 LBL 15
19 LBL 26	44 TONE 8	69 GTO 08	94 LBL 11	119 GTO 20	144 TONE 8
20 TONE 8	45 LBL 16	70 LBL 38	95 TONE 8	120 LBL 44	145 LBL 13
21 LBL 04	46 TONE P	71 XEQ 18	96 GTO 01	121 XEQ 26	146 TONE 8
22 TONE 8	47 GTO 07	72 GTO 09	97 LBL 61	122 GTO 13	147 LBL 20
23 GTO 09	48 LBL 33	73 LBL 53	98 TONE 8	123 LBL 51	148 TONE 8
24 LBL 59	49 XEQ 19	74 TONE P	99 GTO 22	124 XEQ 19	149 LBL 32
25 TONE 8	50 GTO 14	75 LBL 08	100 LBL 36	125 GTO 13	150 END

(305 bytes / SIZE 000)

-Store an alpha string of at most 24 characters in the alpha register and execute "MC"
-Do not use lower case letters.
-Use XTOA to store special characters in alpha (for instance, 64 XTOA adds @ to the alpha string)

Example: Place "HEWLETT PACKARD" in the alpha register , XEQ "MC" and you'll hear:

dih dih dih dit dit dih daah daah dih daah dih dit dit daah daah
dih daah daah dit dih daah daah dih daah dit daah dih daah dih daah dih daah dit daah dih dit

Notes:

-The XEQ IND X (line 09) is much faster if the program is executed from an HEPAX module.
-You can also transmit a message by groups of at most 6 characters, after storing them into contiguous registers Rbb thru Ree (bb > 00

```
LBL "MESSAGE"
STO 00
CLA
LBL 00
ARCL IND 00
XEQ "MC"
ISG 00
GTO 00
END
```

-Place the control number bbb.eee in X-register and execute "MESSAGE"

2°) Learning Morse Code

-This short routine transmits a random message of 1 to 24 characters among 1 2 .... 9 A B C .... Y Z
-You have to decipher the message.

Data Registers: • R00 = seed (initialize R00 before executing LMC)
R01 thru R04 = the characters

Flag: F26

Subroutine: "MC"

01 LBL "LMC"	09 STO 00	17 7	25 ASHF	33 ARCL 01
02 CF 26	10 36	18 +	26 ASTO 02	34 ARCL 02
03 CLA	11 *	19 LBL 01	27 ASHF	35 ARCL 03
04 57	12 INT	20 XTOA	28 ASTO 03	36 ARCL 04
05 LBL 01	13 48	21 RDN	29 ASHF	37 SF 26
06 RCL 00	14 +	22 DSE Y	30 ASTO 04	38 XEQ "MC"
07 R-D	15 X<=Y?	23 GTO 01	31 LBL 10	39 END
08 FRC	16 GTO 01	24 ASTO 01	32 CLA	

(67 bytes / SIZE 005)

-Place a seed in R00 and the number of characters (between 1 & 24) in X-register and XEQ "LMC"
-Press XEQ 10 to repeat the message.

Example: 1 STO 00 4 XEQ "LMC" >>>> dih daah daah dih daah daah daah daah daah daah dit daah dih daah

-The message is AY9K (here in R01)

Reference:

[1] Keith Jarett - "HP-41 Synthetic Programming Made Easy" - Synthetix