Othello for HP-67 & HP-97 (genuine)

Happy birthday to these mythical programmable calculators being 40 this year 2016.

First things first:
No calculator has been harmed during the making (debugging included) of this program :-)
Thank you to all people who helped me with the translation.

40+ year old joke: how do you fit 5 elephants into a Citroën 2CV ? Answer, 2 in the front, 2 in the back and 1 in the trunk.
Now seriously, how do you fit an Othello playing program in the 224 steps of a 1976 pocket calculator ? Well, my answer is on the next page.
This program will play against anyone on a 8 x 8 board, using only 21 memory registers and less than 200 program steps. It will never make the first place in a championship, but the challenge was: can I fit a decent Othello program in a pocket calculator a million times less powerful than our phones 40 years later. All we have here are 224 program steps, 26 registers and 4+1 more from the RPN stack, 4 flags et 3 subprogram levels, all this running at a whopping 30 steps a second.
At the time, it was the height of luxury! If only I had had one…

If you want to reverse engineer my code from scratch for the fun, you will find all you need on page 2, nothing else needed.
If you'd like an overall presentation of what's under the hood, read pages 3 to 5 before going back to page 2.
In short, the more you read on, the easier it is.

It's up to you how you use it. Happy reading!
…and dont forget, with so many program steps left, you can write your own evaluation function : -)

The program code

...OK, this is what we are here for!

Nowadays, we use high-level languages generating many (tens or hundreds) "program steps" for every single line of code we write.
So naturally here we enter into another world. Pure happiness, in my case!

Here, we have less than 200 bytes for the program, and some 21 memory registers of 7 bytes each.
All in all, a reasonable memory footprint.

Some HP-67 and 97 emulators allow for the allocation of hundreds of additional memory registers and thousands of programs steps for good measure.
This would have been a different choice, negating all the fun here.
This program runs on a genuine 67/97 and should work on every emulator that scrupulously reproduces the behavior of both calculators, including their limitations, of course.

There are 25 steps and 10 registers left for improvements. More if you find additional optimisations. So, if you want to customize the code for your 67/97, you have the means!

Please note:
Roll down instructions are written RDN instead of R followed by a down arrow (I was too lazy here)

Instruction mnemonic	HP-97 Key code
*LBL A	21 11
1	01
STO E	35 15
8	08
.	-62
8	08
STO B	35 12
CLX	-51
STO C	35 13
STO D	35 14
*LBL a	21 16 11
CLX	-51
STO 9	35 09
STO A	35 11
RCL B	36 12
CF2	16 22 02
GSB 9	23 09
X=0?	16-43
F3?	16 23 03
GTO 0	22 00
*LBL b	21 16 12
2	02
.	-62
2	02
STO A	35 11
*LBL c	21 16 13
SF0	16 21 00
*LBL d	21 16 14
RCL B	36 12
STO I	35 46
*LBL e	21 16 15
RCL A	36 11
1	01
.	-62
1	01
-	-45
X=0?	16-43
CF0	16 22 00
X=0?	16-43
SF2	16 21 02
RCL I	36 46
+	-55
GSB 9	23 09
F3?	16 23 03
CLX	-51
X=0?	16-43
GTO 0	22 00
X<0?	16-45
GTO 1	22 01
.	-62
1	01
ST+ 9	35-55 09
GTO e	22 16 15
*LBL 1	21 01
F1?	16 23 01
X>0?	16-44
GTO 0	22 00
F0?	16 23 00
X>0?	16-44
GTO 0	22 00
CF0	16 22 00
GTO d	22 16 14
*LBL 0	21 00
RCL 9	36 09
FRC	16 44
ST- 9	35-45 09
1	01
0	00
x	-35
x	-35
ST- 9	35-45 09
RCL A	36 11
.	-62
1	01
-	-45
STO A	35 11
X<0?	16-45
GTO 2	22 02
FRC	16 44
.	-62
2	02
X>Y?	16-34
GTO c	22 16 13
LSTX	16-63
INT	16 34
+	-55
STO A	35 11
GTO c	22 16 13
*LBL 2	21 02
F1?	16 23 01
GTO 3	22 03
RCL B	36 12
+	-55
STO B	35 12
RCL C	36 13
RCL 9	36 09
X≤Y?	16-35
GTO 2	22 02
STO C	35 13
LSTX	16-63
STO D	35 14
*LBL 2	21 02
RCL B	36 12
X>0?	16-44
GTO a	22 16 11
RCL D	36 14
X=0?	16-43
GTO 3	22 03
STO B	35 12
R/S	51
SF1	16 21 01
GTO b	22 16 12
*LBL 3	21 03
RCL E	36 15
0	00
X≤y?	16-36
R/S	51
*LBL B	21 12
CF1	16 22 01
X=0?	16-43
GTO A	22 11
STO B	35 12
SF1	16 21 01
SF0	16 21 00
1	01
CHS	-22
STO E	35 15
GTO a	22 16 11
*LBL 9	21 09
STO I	35 46
SF3	16 21 03
INT	16 34
X=0?	16-43
RTN	24
9	09
X=Y?	16-33
RTN	24
LSTX	16-63
FRC	16 44
1	01
0	00
x	-35
X≠0?	16-42
X=Y?	16-33
RTN	24
CF3	16 22 03
RCL (i)	36 45
X<>Y	-41
4	04
X<>Y	-41
Y^x	31
STO 0	35 00
÷	-24
FRC	16 44
4	04
ST÷ 0	35-24 00
x	-35
INT	16 34
1	01
-	-45
RCL E	36 15
STx 0	35-35 00
x	-35
x<0?	16-45
RTN	24
F1?	16 23 01
F0?	16 23 00
RTN	24
F2?	16 23 02
GTO 9	22 09
X=0?	16-43
RTN	24
SF2	16 21 02
*LBL 9	21 09
RCL 0	36 00
ST- (i)	35-45 45
F2?	16 23 02
ST- (i)	35-45 45
RDN	-31
RTN	24
*LBL E	21 15
8	08
STO I	35 46
2	02
1	01
8	08
4	04
5	05
*LBL 4	21 04
STO (i)	35 45
DSZ I	16 25 46
GTO 4	22 04
1	01
9	09
2	02
R/S	51
ST+ 4	35-55 04
ST- 5	35-45 05
CLX	-51

…Taht's all Folks!

hello [at] jmverniajou [dot] com

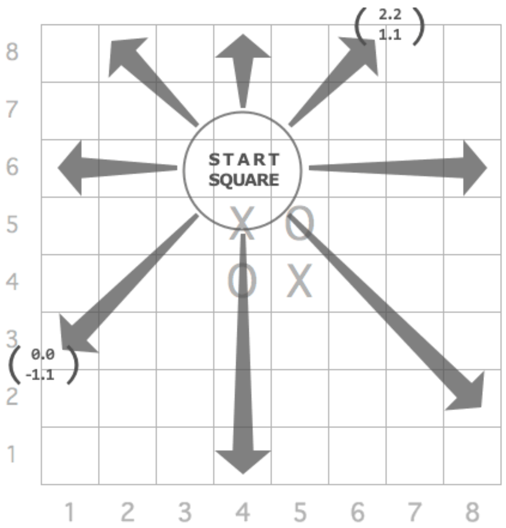
VUE D'ENSEMBLE, TERMINOLOGY AND USER'S GUIDE

To the careful, to the demanding, to the purists, with all my consideration

Having written and commented this program, I decided to change some of the details to get closer to the official Othello game notation i.e. column-row and not row-column,, etc. However, after making the necessary changes, I found the resulting code and comments were less intuitive (if that is possible). So I chose to roll back. You can still make the A1 coordinate top-left, even if the included sample display spreadsheet (see below: displaying the board with a spreadsheet) is showing otherwise.
Side values are hard coded in the program. 1 for the calculator, -1 for it's adversary, but two calculators can play one against the other (see below).
My initial goal was to fit an Othello playing program in the 224 steps available. As a programming exercise, it was a great experience. However, the second stage of optimisation down to 180 steps for the main playing logic (i.e. not including the new game initialization setup) was the cherry on the top.
For the rest, I'm happy to hand over to you.
I can hardly express how much happiness this project has given me. From the start to the end, for both of it's stages. I have a fond thought for all the people at Hewlett-Packard who contributed to the birth of these calculators. I send them my warmest thanks.

In short

- The board in memory is base 4 encoded. The contents of each square can be one of three values: 0 represents a calculator's disc, 1 an empty square, 2 an adversary's disc. These values are hard coded. In order to make two calculators play against each other, discs are placed symmetrically but the same values are used on both.
- The evaluation function is very simple: the more discs won (flipped), the better the score is
- Calculators and emulators: in my experience, the program runs smoothly on both 67 and 97 genuine calculators and on Olivier de Smet's emulator (http://sites.google.com/site/olivier2smet2/hp_projects/hp97). By the way, thank you Olivier.



Terminology

Board	The 8 x 8 playing board
rowNum.colNum	Coordinates of a square, separated by the decimal point. Columns are numbered here because the calculator has no alpha capabilities at all, neither for input nor for output (A to E keys are for other purposes). Example: B6 is entered and displayed 6.2 In this document, discs are represented by O (circles) and X (crosses) for practical reasons, in place of white discs and black discs or vice versa.
Side	Square outside of the board. Square coordinates being in the range 1.1 to 8.8, coordinates having a 0 or a 9 are on the side the board.
start square	Is the square from which an exploration is conducted from. This being either to find the best possible move or the action of adding and flipping discs on the board when the calculator or adversary effectively makes a move.
crossed square	Designates each of the squares explored successively starting from the Start square and in each direction possible.

direction Represents in turn each possible direction in which exploration takes place to find discs that could or should be flipped. The value is in the form delta_row.delta_column, denoted dr.dc, and is in the range 2.2 to 0.0. The value must first have the constant 1.1 subtracted, so it's effective range is from 1.1 to -1.1

central point is the combination between the start square and the nul direction

User's guide

- Only the program needs to be entered in memory. Data initialization is done in LBL E. Anecdote: during testing on a 67, I entered a `X≠0?` program step instead of `X=0?`. The result being subtly different, and subtly false, I thought I had to plan a hunt for a remaining bug…
- If you'd like to reclaim the memory space dedicated to LBL E to make an enhanced version of the program, you'll have to enter the values in the memory registers 1 through 8 by hand. Or you could save the register values on magnetic cards before deleting LBL E. You will need two cards, one for playing the white and one for playing the black and run the LBL E twice, one time with CHS key, one time without (see below).

Starting a new game:

- DSP 1 — Forces the display to show a single decimal digit (this is not mandatory, it just makes the display easier to read)
- E `192` [CHS] R/S — Initialises the board for a new game. When the display shows 192, the calculator waits for you to choose your side. You might choose to press R/S key straight away to make the calculator play with black discs, or press CHS key and then R/S key to reverse the setup and have the calculator play with white discs. This is also the way to make two calculators play against each other. The calculator displays `0.0` when it's ready.
- rowNum.colNum B — To play first, enter your move (the coordinates of the square you want to place a new disc in). The game rules state that black plays first.
- or B — To let the calculator play first. Attention: if you do other calculations in the meantime, be sure to press CLX before pressing B

Continuing the game:

- When the calculator makes it's move, it stops and displays the coordinates of the square it is putting a new disc in (*rowNum.colNum*)
- If it makes a valid move, take note and press R/S to let it update it's in-memory copy of the board. When finished it will stop and display `0.0`
  Attention: the duration of this update can exceed one minute!
- If it passes it's turn, it will display `0.0` Please DON'T press R/S key in this case.
- It's your go. Enter your move *rowNum.colNum* then press R/S. If you pass your turn, leave `0.0` and press R/S
  Note: you are supposed to follow the rules of the game and the calculator does not check if your move is allowed.

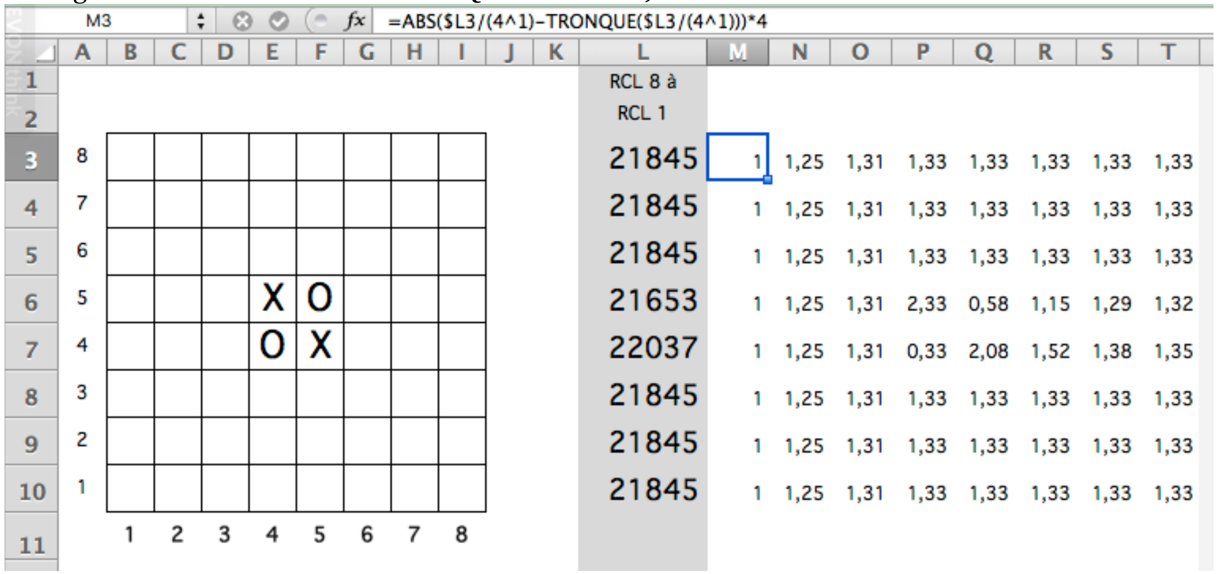
Checking the in-memory copy of the board:

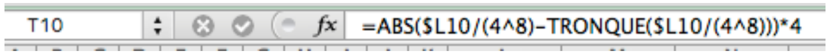
- RCL 1 to RCL 8 (one memory register per row)

hello [at] jmverniajou [dot] com

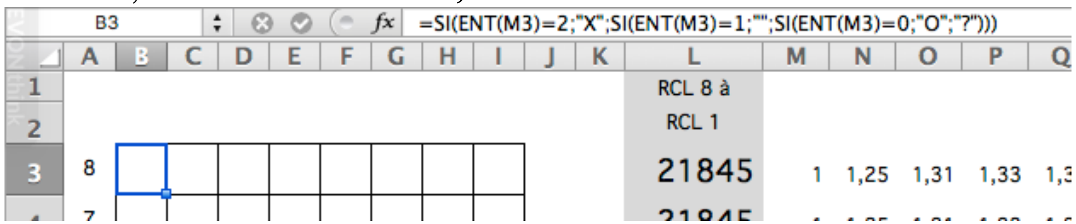
Sample spreadsheet for displaying the in-memory base 4 encoded board:

- Cells M3 to T10 (please note: the English name of the function TRONQUE is TRUNC)





- Cells B3 to I10 (please note: English for SI is IF, and ENT translates to INT)



Note: formulas showed will always display Os for the calculator and Xs for the adversary. These are my notation choices. Feel free to find customizations that will fit your taste.

Warning…

…if you carry on reading,
you will find it easier   :-)

Memory registers allocation

0	temporary calculations
1 to 8	8 x 8 Othello board, base 4 encoded
9	Score for the move being evaluated (the "start square"), expressed in one of these two forms:
	p : integer representing the count of all found flippable discs in the directions already explored
	p.h where h represents the additional hypothetical gain (number of discs) for the direction currently under exploration.
10 à 19	unused
A	Current Direction: in the range `2.2` to `0.0` representing delta_x_+_1.delta_y_+_1 for the rows, columns and diagonals from the Start square
B	Start square coordinates - Either decremented from `8.8` to `1.1` (in fact `0.0`) in the form rowNum.colNum to explore all possible moves - Or the coordinates of the square where a disc is added (by the calculator or adversary) to update the in-memory copy of the board
C and D	Best score found from the possible moves examined so far, and the coordinates of that best move
E	1 means the calculator is the actual player, -1 means the adversary is the actual player

Flags f0 to f3 allocation, in order of importance

Command-cleared flags:

- f1 — cleared=EVALUATION (search of possible moves for example), set=ACT (on the board by adding and by flipping discs)
- f0 — Complimentary to flag 1 to perform an action on the board in two passes. cleared=NOW (2nd pass), set= LATER (postpone changes, 1st pass)

Test-cleared flags:

- f2 — Central point indicator OR add/flip disc selector
- f3 — Side indicator (coordinates are outside the board)

Some essential points concerning the operating characteristics of the HP-67 and 97 calculators

Branches — A simple branch instruction (example GTO 9) transfers execution to the next occurrence of the label, starting from the step following the current step. If such a label can not be found before step number 224 then the search for the label starts over from step `001`.
This gives the opportunity to use **multiple occurrences of the same label number**
In the commented code, these multiple occurrences are suffixed to ease reading. Example: LBL 9, LBL 9 bis, etc.
Also note that upper and lower case characters are differentiated : LBL A and LBL a are completely different

Flags — Besides the peculiarity of flags 2 and 3 being automatically cleared when tested, in addition, there is no "if flag clear" instruction for any of the flags. The only way to test flags is the instruction "if flag set". This leads to a few acrobatic conditional execution arrangements in the program.

RPN stack — Conventional names are used in the code and comments for the registers X, Y, Z et T (for Top). and "Last X".

indirect addressing — The register dedicated to indirect addressing, the only "pointer" available, is named I (I as in… Indirect).
- noted I in the code, for example STO I, the I register itself is involved (we're manipulating the pointer)
- noted (i), means the register involved is the one whose number is stored in I. In this program, indirect addressing is not used for branches, only for manipulating memory registers.

COMMENTED CODE

The essence and the charm of the exercise

The code contains many optimizations required to save program steps so that the whole program can fit available memory space. Also note that the instruction set and the memory dispatching of these calculators are suited for calculations and are less appropriate for other uses such as games. Some tricks were needed to overcome the imposed restrictions.
These optimizations and tricks show in the code as some of the "worst programming practices". Or not ;-)

- In this spirit, LBL 9 runs two threads in parallel:
  The first is making calculations to extract the contents of a given square (8 squares are encoded in each of the 8 registers storing one of the 8 rows of the board), the second builds the appropriate value needed to add or flip a disc in that same square, in the event it will become useful later on.
  It is "la loi du genre": when some data is at hand, the maximum should be done with it to save program steps.
  The comments related to the second thread are grayed out to make it easier to follow each thread separately.
- Nearly all "end if" mentions are omitted. Combined conditional execution is usually commented and indentation in the comments should make the missing end ifs obvious. I hope.
- The key codes listed are those from an HP-97, the printing model of the HP-67
- The Roll down instructions are written RDN not R followed by an arrow pointed downward (sorry, I was too lazy here)

As a final word, I'd say that writing the comments accompanying the code is not always the most interesting part of the adventure and that commenting code across optimizations sometimes impairs readability. Moreover, English is not my native language and some terms I chose might not be the best fits. Please, forgive any subpar descriptions. And of course, any comments or suggestions will be appreciated.

```
 instruction     HP-97   Comments
   mnemonics key codes
  ──────────  ───────-   ──────────────────────────────────────────────────────────────────────────────────────-
      *LBL A     21 11   Board level (it is calculator’s turn)
           1        01             E <- calculator’s turn
       STO E     35 15
           8        08             start square <- the search for possible moves begins at square 8.8 down to 1.1
           .       -62
           8        08
       STO B     35 12
         CLX       -51             zeroing best move
       STO C     35 13                  score
       STO D     35 14                  and coordinates
```

```
*LBL a   21 16 11   ---  start square level
  CLX         -51             zeroing
 STO 9      35 09                  score (cumulative wins obtained by adding a disc in start square)
 STO A      35 11                  current exploration direction
 RCL B      36 12             start square (or coordinates for the move the board is updated for)
  CF2    16 22 02             central point indicator <- false (Direction - 1.1 = -1.1 not 0)
 GSB 9      23 09             pre-examination (coordinates inside/outside board ? empty square ?)
                                   with an exploration direction set to 0 because:
                                        - if appropriate, artificially triggers a skip to the next start square at the
                                        time the skip to the next Direction is triggered
                                        - and by the way, replace GTO 2 below by GTO 0, the code in LBL 0 becomes transparent
                                        in this particular condition, except leaving the value -0.1 on the stack, thus
                                        saving program steps (sign, decimal point and digit 1 not duplicated)
  X=0?      16-43             optimized code replacing "X≠0? GTO 0 F3? GTO 0" I mean:
   F3?   16 23 03             if the square is not empty OR coordinates are on a side of the board
 GTO 0      22 00                  end exploration for this start square (GTO 0 not GTO 2 as explained before)
                              Attention: it is not a true replacement for the original code "F3? GTO 0 x≠0? GTO 0", that
                              cleared the flag f3 by the way in all cases and made the x≠0? test consistent.
                              So, why retain the change (saving one program step) ?
                              - This particular optimization without swapping the tests is not possible because it is not
                              possible to invert test F3? (no if flag clear instruction).
                              - in case a side is hit, X can not represent the contents of the square (the square does not exist)
                              and the test X=0? is either adequate by chance, or transparent except for f3 staying set.
                              - in the case the square is inside the board, X holds the contents of the square. if it is not
                              0 because the square is not empty, execution effectively GTO 0, else the flag f3 not being set,
                              the execution goes on with LBL b because GTO 0 is stepped over

*LBL b   21 16 12        OK, It seems that start square is a good candidate to add a disc
     2         02             Initialize exploration Direction to 2.2
     .        -62                  (will successively be 2.1 then 2.0, 1.2, 1.1, 1.0, 0.2, 0.1 et 0.0 values from which,
     2         02                  by subtracting 1.1, we get the "delta_x.delta_y" in the 1.1 to -1.1 range
 STO A      35 11                  that will be added repeatedly to obtain the coordinates for each crossed square following
                                   each exploration direction: row, column and diagonal

*LBL c   21 16 13   ---  Direction level
   SF0   16 21 00             f0 <- LATER (in case, action on the board begins by first pass)

*LBL d   21 16 14             Direction exploration always starts at start square
 RCL B      36 12                  Copy the coordinates of start square in the register
 STO I      35 46                  dedicated to indirect addressing. It will point toward crossed squares
```

```
*LBL e   21 16 15   —-  Crossed square level (going on with exploration for that direction with the next crossed square)
 RCL A      36 11             Direction in the range 2.2 to 0.0
     1         01                  1.1 = central point
     .        -62
     1         01
     -        -45             dr.dc <- Direction - 1.1 (now in the range 1.1 to -1.1)
  X=0?      16-43             if the Direction dr.dc is 0.0, it is not a "real" direction, it is the central point
   CF0   16 22 00                  act NOW
  X=0?      16-43                  with
   SF2   16 21 02                  central point indicator <- true
 RCL I      36 46             crossed square coordinates <- previous crossed square coordinates + dr.dc
     +        -55
 GSB 9      23 09             examination of the coordinates of the crossed square and its contents
   F3?   16 23 03             if the crossed square is on a side
   CLX        -51                  simulate the encounter of an empty square
  X=0?      16-43             if the crossed square is empty
 GTO 0      22 00                  end exploration for this Direction with cancelation of hypothetically won discs, etc.
  X<0?      16-45             if the square contents is a disc owned by the actual player
 GTO 1      22 01                  switch to 2nd pass or end exploration for this Direction (etc.) depending on context
     .        -62             else the square contains a disc belonging to the other player
     1         01                  add 0.1 to the score (means 1 disc hypothetically won in that direction)
 ST+ 9   35-55 09                  the evaluation function is very simple: "the more I flip discs, the better"
 GTO e   22 16 15             Continue with this Direction with the next crossed square

*LBL 1      21 01       What to do when Direction exploration ends ? (next lines of code: remember there is no if flag clear…)
   F1?   16 23 01             if ACT
  X>0?      16-44                  test whose result is always false to compensate for the missing of if flag clear
 GTO 0      22 00             so, is equivalent to If EVALUATION then amalgamate the score and program the next Direction
   F0?   16 23 00             if LATER (end of 1st pass)
  X>0?      16-44                  test whose result is always false to compensate for the missing of if flag clear
 GTO 0      22 00             so, is equivalent to If NOW (end of 2nd pass) then amalgamate the score etc.
   CF0   16 22 00             else      NOW (end of 1st pass, switch to 2nd pass)
 GTO d   22 16 14                       and redo Direction from Start square to effectively flip the discs

*LBL 0      21 00   Amalgamate the score and program the next Direction
 RCL 9      36 09       The count of hypothetically won discs for the current Direction is stored
   FRC      16 44             in the decimal part of the register holding the score and now that we have it
 ST- 9   35-45 09             we can clean up the score register to its integer value before that direction exploration
     1         01       This count needs to be
     0         00             transformed into an integer
     x        -35             to be added to the score, but also
```

```
          x       -35       multiplied by the input value (contents of the square or equivalent), remember:
                                 0 (no discs to flip)
                                 - if exploration of Direction ends on a side of the board
                                 - if exploration of Direction ends with an empty square
                                 9 only after a pre-examination, when the hypothetical gain is 0
                                 - if exploration of Direction ends on a side of the board
                                 or -1 (discs to flip)
                                 - if exploration of Direction ends with a square whose disc belongs to the actual player
      ST- 9  35-45 09       … for a final result: the number of discs actually flipped/flippable (a negative number)
      RCL A     36 11       Programing the next Direction
          .       -62            2.2 -> 2.1 -> 2.0 -> 1.2 etc.
          1        01                 dr.dc is used with rowNum.colNum
          -       -45                 by simply adding both values and compensating the 1.1 offset
      STO A     35 11            next direction
       X<0?     16-45            if next Direction is below 0 (all Directions have already been explored for this start square),
      GTO 2     22 02                 exploration for this start square is over, we leave -0.1 in X
        FRC     16 44            else exploration for this start square is to be continued,
          .       -62                      in the condition to adjust, when appropriate, the new Direction value to stay within
          2        02                      authorized values (2.2 to 2.0 or 1.2 to 1.0 or 0.2 to 0.0)
       X>Y?     16-34                 if the decimal part is below 0.2, the value for the Direction is OK
      GTO c  22 16 13                      go explore that new Direction (run 67, run!)
       LSTX     16-63                 else
        INT     16 34                      the correct value for the Direction is obtained by adding to the integer part (0 ou 1)
          +       -55                      the value 0.2
      STO A     35 11                      now, the next direction is OK
      GTO c  22 16 13                      go explore that new Direction

     *LBL 2     21 02  End of exploration for this start square (all directions have been followed)
        F1?  16 23 01       if ACT
      GTO 3     22 03            we just added and flipped discs on the board, whose turn is it ?
      RCL B     36 12       else
          +       -55            add the -0.1 value left on the stack to the coordinates of the start square
      STO B     35 12            to set the coordinates of the new start square (8.8 -> 8.7 -> … -> 0.0 -> -0.1)
      RCL C     36 13       Compare the best score for the best move so far
      RCL 9     36 09       with the score for the start square we just examined
       X≤Y?     16-35       if it is not better
  GTO 2 bis     22 02            skip the following program steps that store the new best move
      STO C     35 13       else remember the new best score
       LSTX     16-63            and the coordinates of the start square (for which we just closed evaluation)
      STO D     35 14            as the new best move
 *LBL 2 bis     21 02  Next Start square or end the board ?
```

```
  RCL B     36 12       if the coordinates of the new start square
   X>0?     16-44       are over 0.0, we're not done yet with all those squares from 8.8 (plethora isn't it?)
  GTO a  22 16 11            Goes on with the new start square (Life is a box of chocolates)
  RCL D     36 14       else displays the best move
   X=0?     16-43            If the calculator passes its turn
  GTO 3     22 03                 no update of the board is needed
  STO B     35 12            else 1) copies the best move coordinates to start square to prepare for the board update
    R/S        51                 2) then stops to let the adversary take note of the move
    SF1  16 21 01                 3) chooses ACT
  GTO b  22 16 12                 4) and updates the board

 *LBL 3     21 03  Switch between the calculator and adversary - part 1 - should the calculator stop to let adversary play ?
  RCL E     36 15       Who is playing right now (-1 = adversary, 1 = calculator) ?
      0        00       displays 0 whatever (signals to the adversary that it is its turn, etc.)
   X≤y?     16-36       if the calculator is playing
    R/S        51            stop execution and display 0 waiting for the adversary to enter its move and press R/S key
 *LBL B     21 12  Switch between the calculator and adversary - part 2 - if adversary passes, its calculator's turn
    CF1  16 22 01       EVALUATION (in the event the adversary passes its turn)
   X=0?     16-43       if adversary passes its turn
  GTO A     22 11            it is calculator's turn so execution starts over with the fetch for a new move
  STO B     35 12  Switch between the calculator and adversary - part 3 - start square <- the move of the adversary
    SF1  16 21 01            ACT to update the board, adding and flipping discs accordingly
    SF0  16 21 00            LATER (1st pass) helps, during pre-examination, to detect a move in a non-empty square
      1        01
    CHS       -22
  STO E     35 15            E <- –1 (prepares to act on behalf of the adversary)
  GTO a  22 16 11            add and flip discs on the board (will automatically go on with the fetch for the next move)

 *LBL 9     21 09  What is possible with these coordinates ?
  STO I     35 46       pointer <- rowNum.colNum
    SF3  16 21 03       First, detect the sides of the board
    INT     16 34            rowNum
   X=0?     16-43            if rowNum = 0
    RTN        24                 return, out (bottom side)
      9        09
   X=Y?     16-33            if rowNum = 9
    RTN        24                 return, out (top side)
   LSTX     16-63            rowNum.colNum
    FRC     16 44            colNum is in the decimal part
      1        01
      0        00
```

```
         x       -35          colNum
     X≠0?     16-42          optimization: if not left side (the initial code was x=0? RTN x=y? RTN)
     X=Y?     16-33          test for right side
      RTN        24          return if out (left or right side) else goes on
      CF3  16 22 03     rowNum and colNum are both inside the board
  RCL (i)     36 45          contents of the row rowNum (stack XYZT: row rowNum, colNum…)
     X<>Y       -41          Isolating the square colNum (stack XYZT: colNum,row rowNum…)
        4        04               the board is base 4 encoded (stack XYZT: 4, colNum,row rowNum)
     X<>Y       -41          (stack XYZT: colNum, 4, row rowNum…)
       Yx        31          (stack XYZT: 4^colNum, row rowNum…)
    STO 0     35 00          r0 <- 4^colNum = building the value to be used later for disc addition/flipping in this square
        ÷       -24          row rowNum / 4^colNum
      FRC     16 44          strips the squares on the left, contents of square colNum is just after the decimal point
        4        04
    ST÷ 0  35-24 00          r0 <- 4^(colNum-1) = the value to be used in fine, but still unsigned
        x       -35          contents of the square colNum is now before decimal point, in the range 0 to 2 (1 = empty square)
      INT     16 34          strips the squares on the right -> X = contents of square rowNum.colNum, finally
        1        01
        -       -45          contents of the square rowNum.colNum in the range -1 to +1 (0 = empty square)
    RCL E     36 15          adjusting with who is the player, the calculator or the adversary
    STx 0  35-35 00          r0 <- 4^(colNum-1) signed = final value to be used later for disc addition/flipping in this square
        x       -35          to find out if the disc belongs to the one the action is done for
     x<0?     16-45          if the disc belongs to the actual player
      RTN        24               return
      F1?  16 23 01          else      optimized code for:
      F0?  16 23 00                    if EVALUATION or LATER (1st pass)
      RTN        24                          return
                        Action on the board now has to be taken
                             To sum-up:
                                  if it is the central point, a disc needs be added for the actual player
                                  else, if the square is empty, this is the end of the explored Direction
                                  else, the disc must be flipped
                                  Attention:
                                       in case of a pre-examination (from LBL a), the central point indicator is not
                                       set as such

      F2?  16 23 02          if central point indicator is set
GTO 9 bis     22 09               will add a disc on the board
                                  and clears the flag 2 by the way,
                                  so f2 can be reused because it is cleared whatever, I love it
     X=0?     16-43          else, if the square is empty
```

```
        RTN         24                return (end of exploration for this direction)
        SF2   16 21 02           else the next thing to be done on the board will be a disc flip
 *LBL 9 bis      21 09      Adding or flipping a disc in square rowNum.colNum
      RCL 0      36 00           signed value for action in colNum ( x 1 to add disc, x 2 to flip disc)
    ST- (i)   35-45 45           modifying the contents of the square (adding a disc) in the row
        F2?   16 23 02           if action is flipping
    ST- (i)   35-45 45                complete the action on the contents of the square (flipping a disc) in the row
        RDN        -31      recalls the contents of the square before the disc was added or flipped
        RTN         24      return

     *LBL E      21 15  Board initialization for a new game
          8         08      8 rows to initialize
      STO I      35 46
          2         02      an empty row is encoded 21845
          1         01           ((((((1x4+1)x4+1)x4+1)x4+1)x4+1)x4+1)x4+1
          8         08           you’re right, this is not Reverse Polish Notation :-)
          4         04
          5         05
     *LBL 4      21 04      For each memory register in the range 8 to 1
    STO (i)      35 45           Initialize the row with the value for an empty one
      DSZ I   16 25 46           i <- i - 1 and while i > 0…
      GTO 4      22 04      …next row please
          1         01      Adding X and O discs to start the game
          9         09
          2         02
        R/S         51      Waits for the adversary to choose his(her) side
                                 - presses strait on the R/S key to let the calculator play with the black discs
                                 - presses CHS key then R/S key to let the calculator play with the white discs
      ST+ 4   35-55 04      OX ou XO added in row 4
      ST- 5   35-45 05      XO ou OX added in row 5
        CLX        -51  Ready!
```