ILLUMINATI II: THE DAVOS DEADLOCK

HOW THE CODE WORKS

ACHTUNG!

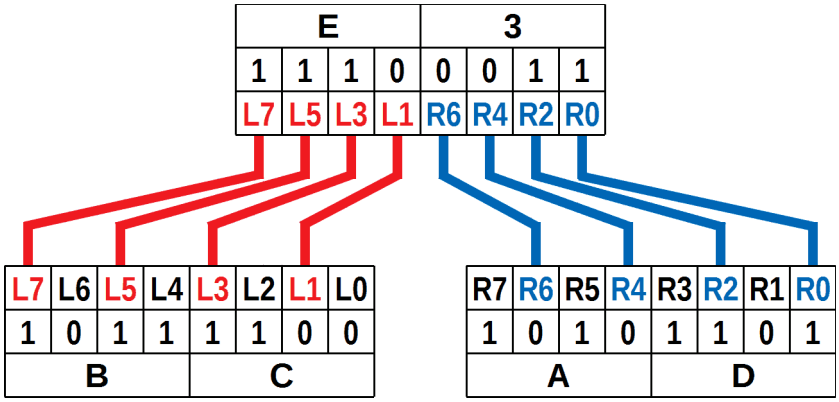
I put all the .ASM files in the package with the game, so everything you'd need to know is all there, spelled out in heavily-annotated form, with what just about every instruction does, unless it's so blindingly obvious it wasn't worth mentioning.

Hence, looking any further through this document is an intrinsic admission that you are either too lazy to disassemble and disentangle the machine code yourself... or incapable of understanding it.

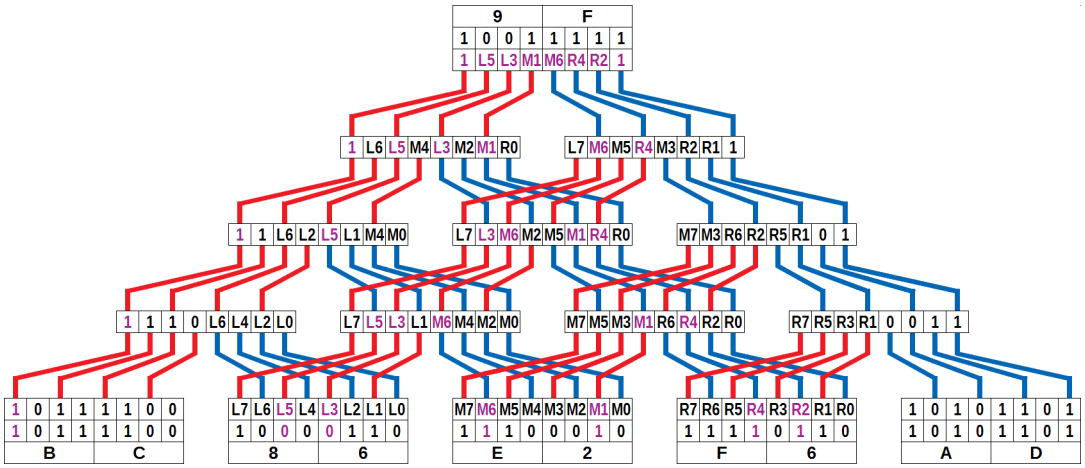
Either way, this does not reflect well on you.

If you're OK with that, then continue.

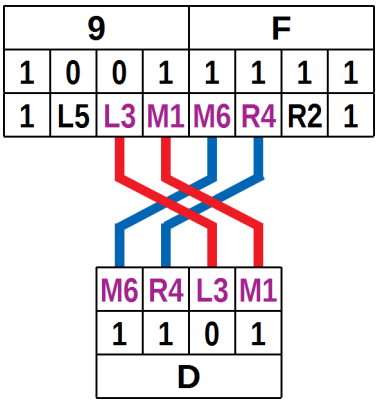
This is how the original Illuminati code worked; whether it was by brute force, i.e. the BIT and SET instructions on the ZX81, or the more sophisticated method of the Spectrum involving rotate and shift instructions, the result was the same. Bits 7, 5, 3, 1 of the left hex byte are concatenated with bits 6, 4, 2, 0 of the right hex byte. Thus, BC on the left and AD on the right results in E3. This is the standard test when rewriting the code in other assembly languages to see if it works correctly.



Because of the BC and AD at the bottom left and right of the pyramid, bits 7 and 0 of the final result would always be 1, so the first hex digit of the answer was always 8-F and the second hex digit was always odd (1, 3, 5, 7, 9, B, D, F).



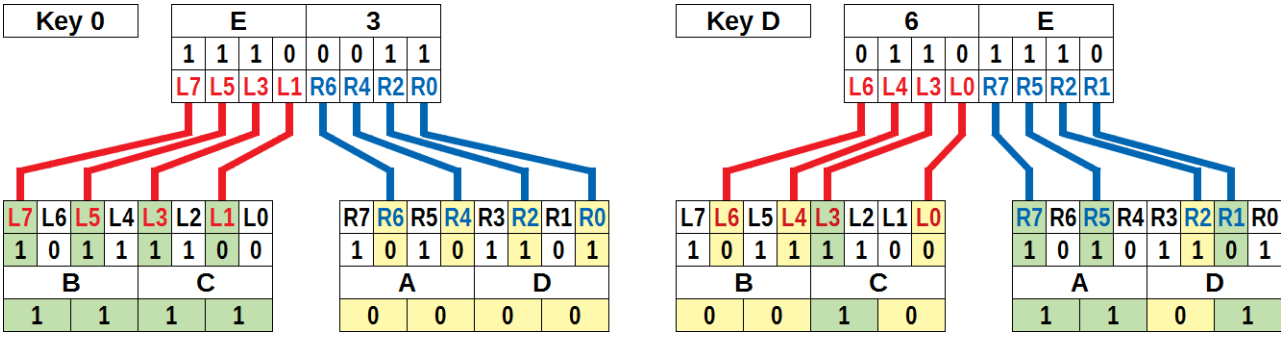
In the original game, that was the end of that. In Illuminati II, this first result is used to generate the key for the second pass:



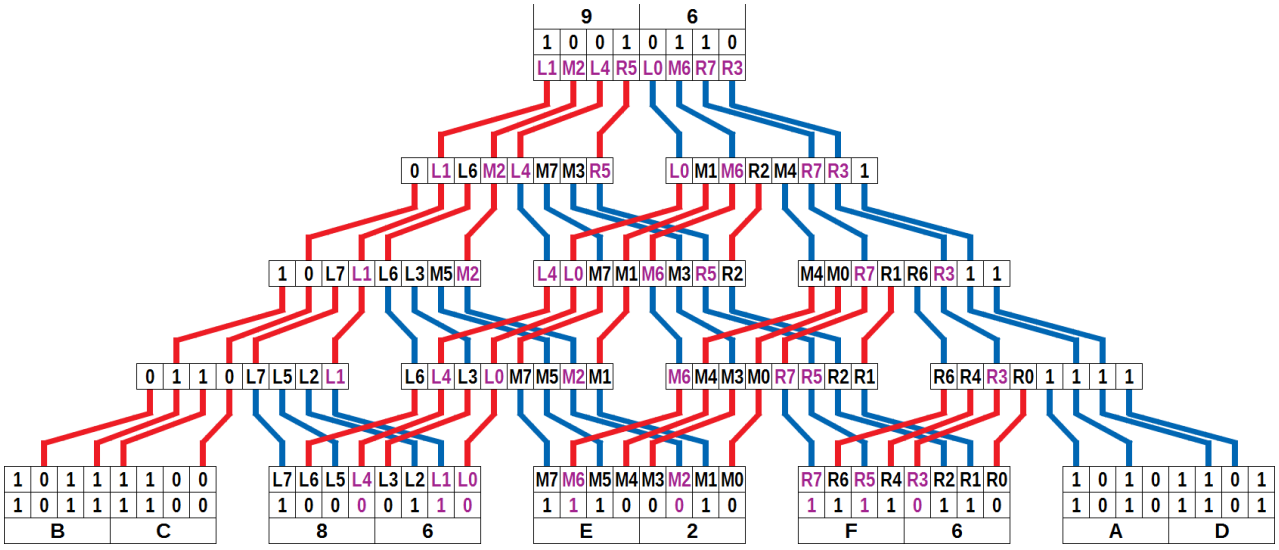
To see how the key works, first split each of the hex bytes into pairs of bits – 7/6, 5/4, 3/2 and 1/0.

Now view the key in its binary representation – 0000 for key 0, which is the original code, 1101 for key D which has been generated by the example pyramid in the first pass.

For each of the four bits in the key: 0 means select the lower bit of the pair in the **right** byte, and 1 means select the higher bit of the pair. The **opposite** is applied to the **left** byte, i.e. the 4-bit complement of the key is used (i.e. F for key 0, 2 for key D).



Hence the test pair of BC and AD now produces 6E instead of E3 – and the second pass through the pyramid with key D instead of key 0 produces a different result:



Using key D, it can be seen that any final value is possible – it is not constrained to odd numbers above 80 as the key does not force a 1 into bits 7 and 0 of the result.

It should also be possible to see how this new method blocks the possibility of reverse engineering a starting combination from the result 96 – without knowing which key has been used, it is not possible to trace a route back to the highlighted starting bits. And, in contrast to the original code, the other bits cannot be filled in at random, as they are needed to generate the key in the first place; in this example, only bit M6 is involved in both the key generation and the final result.