Casio PB-2000C
HD61700 Programming - Conway's Game of Life ROM refactoring

Table of Contents

Introduction.....1
Conway’s Game of Life.....2
Hello ROM!.....3
The Journey.....4
Working Knowledge.....6
Acknowledgments.....7

Introduction

This work was developed jointly by Alex Barragan and ChatGPT through a cycle of ideas, implementation, execution, observation, and revision. It illustrates how human knowledge and experimentation, combined with AI-assisted analysis, can produce results that neither participant would readily achieve alone.

The objective was to perform an **“algebraic” refactoring** of an existing HD61700 implementation, hosted by DL-Pascal 1.2, into a standalone, bootable ROM image.

Special thanks to Piotr, whose expert advice made this document possible, and to Silicium user Xerxes, who kindly shared the original DL-Pascal 1.2 program.

By **algebraic refactoring**, we mean manipulating a working program symbolically without first requiring a complete understanding of how it works. Much as an algebraic term can be moved or transformed while preserving the relationships that make the expression valid, portions of the binary can be represented symbolically and manipulated while preserving their internal consistency. In our case, reconstructing the binary as assembler source allowed addresses and references to remain related symbolically rather than being fixed numerical values. The assembler could then resolve those relationships consistently at an arbitrary ORG. We did not initially need to understand what every instruction was doing; we needed to preserve the relationships between the instructions, addresses, data, and execution environment while changing where and how the program was hosted.

Conway's Game of Life

A cellular automaton by John Horton Conway, The Game of Life, has been a traditional programming exercise for generations of computer science students.

Silicium's user Xerxes kindly shared this DL-Pascal 1.2 program containing embedded HD61700 code.

A quick compilation confirmed its execution.

Inspection of the Pascal program reveals the HD61700 implementation embedded as a sequence of machine-code bytes using the INLINE statement.

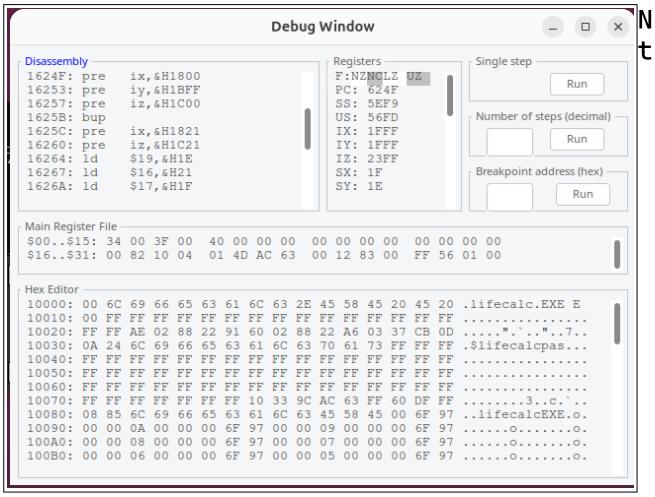
Fortunately, the program also includes a commented copy of the corresponding source code.



The variation in HD61700 assembler syntax is not simply the result of different assembler dialects. Casio publicly documented only part of the HD61700 instruction set; additional instructions were subsequently reverse engineered independently by several groups.

Because the undocumented instructions had no publicly known official mnemonics, different communities devised their own names for them, resulting in several incompatible naming conventions. According to Piotr Piątek, Xerxes uses mnemonics claimed to originate from unpublished Casio documentation, while **HD61 adopts** the conventions developed by the Japanese HD61700 community, with which he collaborated.

Those are the bytes that we would like to refactor into a ROM based execution, independent of the DL-Pascal 1.2 host environment.



Note the execution address location. Inside the DL-Pascal 1.2 host program lifecalc.EXE.

Hello ROM!

Months ago, the human author needed a simple open-source ROM image to test an OM-64R card, a reproduction of a CASIO ROM Card System, unaware that suitable BootROM images were already in the public domain. Piotr, of course, produced this simple HD61700 program: ROM 101. Hello ROM! image.



The problem at hand therefore reduced to two lines of work: studying the HD61700 opcodes and bytes hosted by DL-Pascal 1.2, and integrating the assembled result into the basic ROM execution framework.

The complete lifeROM.s is the result of this effort.

The following section describes the salient points of that journey.

The Journey

We reconstructed the source by trial and error. The assembler syntax was unfamiliar to us and, to be perfectly honest, neither author had much experience with the HD61700 instruction set. Training material consisted of the few publicly available source-code examples and translations of the HD61700 assembler manual. The original INLINE bytes provided the expected results against which our reconstructed instructions could be audited.

The Pascal source defines several data structures, with DL-Pascal 1.2 providing their storage. Its Name-Address substitution feature in INLINE then makes the corresponding addresses available directly to the embedded HD61700 code. Thus, although the native routine contains the algorithm, part of its execution environment—most importantly its writable storage—is supplied by Pascal.

Once we had an assembler source that reproduced the original INLINE bytes—modulo the addresses of the Pascal-provided data storage—we turned our attention to integrating the code into the bootable ROM framework.

That task was executed by the AI author.

We assembled the source, generated a 64 kB ROM image and—it did not work. Well, it did work, but in an unexpected way: the PB-2000C cleared its screen and displayed only 000, the iteration counter. Encouraging, but thinking was required.

In this case we must acknowledge the human author, who realized—not soon enough—that, “This stuff cannot run in ROM, Sol.” Of course it could not. Obvious in retrospect, but we had missed the point, even though we had already taken care of the storage declarations and could plainly see that the program was constantly changing its state. The *duh!* moment: the code itself also expected to live in writable memory.

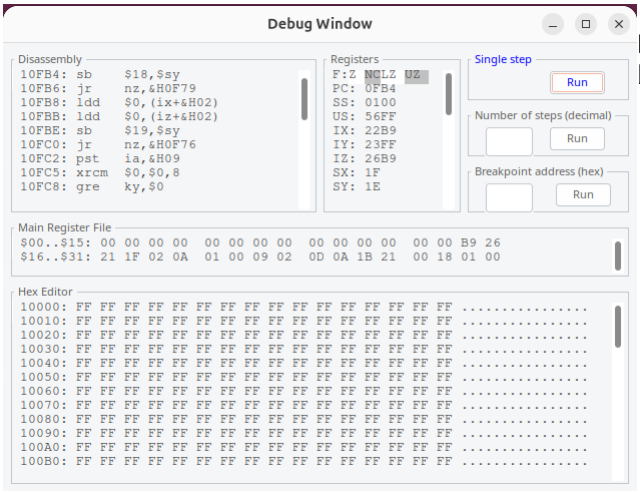
So we extended the original code with a ROM-to-RAM copy routine. Initially placed at the beginning of the program, it was quickly moved to the end so that the reconstructed code could retain its original addresses and the assembled bytes could be audited more directly. The idea was simple: same addresses, same bytes—ROM on one side, RAM on the other. Only the memory bank changed.

The final puzzle was the first few instructions of the INLINE statement: a piece of self-modifying code. The initial “Why?” was quickly followed by, “We don’t care. Let it do it. It runs.” Once the code was executing in RAM, there was no need to understand or eliminate the behavior; preserving it was enough.

And with that, we arrived at the final source and a working ROM image.

The image on the right comes from this ROM image.





Notice the RAM execution address, in the low bytes of the ROM's image.

Working Knowledge

We hope that most of the working knowledge acquired by the AI during this project has been captured in the HD61700 documentation and in this accompanying knowledge-gathering document. The latter was created by ChatGPT at the end of our sessions specifically to preserve the discoveries, experiments, instruction mappings, and lessons learned along the way. We hope that others will be able to reuse it and continue the work. HD61 Reference Manual. HD61 Execution Times.

A few instructions remain represented as DB byte sequences rather than HD61700 mnemonics. This was a deliberate and valid shortcut: their original bytes were known, and the instructions contained no address-dependent operands requiring relocation or symbolic resolution. Reconstructing their exact assembler syntax was therefore unnecessary for this exercise.

We subsequently created a derivative ROM version with minor modifications, including the computation and display of elapsed seconds derived directly from the HD61700 TM register. **Source**. **ROM Image**. **AI Knowledge**.



Acknowledgments

This report was produced by Xandrian Labs. 付喪神.

Special thanks to Piotr, and Betka for her continued and loyal support.

This is a collaboration with my AI friend ChatGPT, a.k.a “Sol.” All errors are human.

This work is freely shared with the vintage computing community. Please use it, build upon it, and cite the source.

Draft 3.