Microcode-Level Calculator Simulation

Eric Smith

Many people want capabilities equivalent to HP calculators on other devices such as desktop computers or PDAs. This need is often addressed by functional level simulation, i.e., programs which attempt to provide the user-visible behavior of the calculator without modeling the exact internal state of the real calculator's hardware and microcode. For instance, there was an HP-25 functional simulator which ran on the Apple II computer in the early 1980s. HP has at times offered functional-level calculator simulators commercially, including an HP-12C financial calculator simulator in the HP-95 series of palmtop computers, and HP-11C scientific, HP-12C financial, and HP-16C computer science calculator simulators in the HP-UX desktop environment.

Functional-level simulation may be adequate for many user needs, but generally does not produce numerical results identical to those of the real calculator. For instance, the arithmetic precision may be different, and the arithmetic may even be performed in a different base. The HP-UX calculators use IEEE 754 binary floating point, while the original calculators use BCD floating point.

The mathematical algorithms also may be different. For simple operations this is typically not significant, but for complex operations like interest rate calculations, HP invested considerable resources into numerical analysis and algorithm development to make the results as accurate as possible given the limited precision and memory size of the calculator, and functional simulators often use inferior algorithms yielding worse results.

Functional simulators also often differ from real calculators in handling exceptional conditions and edge cases. An extreme example of this is the inability of most HP-41C functional simulators to fully handle synthetic programming[1].

It is desirable to have an alternative simulation methodology that is more nearly identical to the real calculator in both numerical accuracy and functionality. This can be achieved through microcode-level simulation.

1 Synthetic programming is the use of unintended characteristics of the calculator to obtain behavior or results impossible or difficult to obtain through normal means. Often this is done by taking advantage of bugs. The term is most commonly associated with the HP-41C [Wickes1980].

History of Microprogramming

The earliest stored-program computers had a hardwired instruction set. Memory was expensive, so it was desirable for the instructions to perform fairly high-level operations to maximize the program density. However, the amount of logic and wiring necessary to implement complex instructions is significant. In 1951, M.V. Wilkes proposed a technique known as microcoding, in which the programmer-visible instructions of a computer are themselves implemented by the execution of a microprogram on a simpler machine [Wilkes1951].

Microprogramming first gained wide acceptance with its use in the IBM System/360 series of mainframe computers, introduced in 1961. In addition to using microcode to implement the System/360 instruction set, these machines pioneered the concept of "emulation", which IBM defined to be the use of dedicated hardware and/or microcode to simulate an otherwise incompatible computer system. An option was offered for the IBM 360 Model 30 to emulate the earlier IBM 1401 computer.

In this paper the term "simulate" is used rather than "emulate" with regard to calculator simulation since no dedicated hardware or microcode is used on the simulation host system.

Microprogramming in HP Calculators

All HP desktop and handheld calculators are microprogrammed. The microprogram is responsible for the basic operation of the machine, floating point number entry, display, and arithmetic, transcendental functions, user program entry and execution, peripheral control, etc.

In order to implement floating point addition, the calculator must execute a series of microcode instructions to compare the exponents of the two operands, shift one and increment or decrement its exponent until the exponents match, add the mantissas, normalize the result, deal with the signs, etc. The processor provides BCD integer operations on subfields of the registers, and the necessary comparison and conditional branch instructions. For instance, once the exponents are equal and the signs have been dealt with, the actual addition of the mantissas is accomplished by an instruction like "c = c + a [m]", where "a" and "c" are working registers, and "[m]" specifies that the operation is to be performed on the mantissa field (digits 2 through 12, numbered from right to left).

HP first used microprogramming in the HP 9100A desktop programmable scientific calculator [Cochran1968], introduced in 1968, though the designers were not familiar with the terms "microprogram" or "microinstruction" [Osborne1982].

HP introduced the first handheld scientific calculator, the HP-35, in 1972. The microarchitecture of the HP-35 was an entirely new design based on different engineering tradeoffs resulting from the availability of high-density PMOS logic circuits and ROMs. A strict Harvard architecture was used, in which there are separate microinstruction and data memories. The microinstruction words are ten bits wide, while data words are 56 bits wide.

With minor changes, the microarchitecure of the HP-35 was used for several succeeding generations of HP calculators, culminating in the “Nut” processor used in the HP-41C family of expandable scientific programmable calculators, and the Voyager series of scientific, financial, and computer science calculators (e.g., HP-11C, HP-12C, HP-16C).

A new processor architecture, “Saturn”, was introduced in the HP-71B handheld computer in 1984 [Dickie84], and was used in all HP handheld calculators introduced from 1986 through 1998, notably including the HP-28 and HP-48 families. Saturn is a Von Neumann architecture, such that there is a single unified address space, and the data word width is increased to 64 bits. Instructions are of variable length in multiples of four bits, the unit of addressing. Nevertheless, many similarities to the earlier calculator microarchitectures were preserved, such as the ability of arithmetic and logical instructions to operate on digit range subsets of a data word.

User Microcode for HP Calculators

HP did not originally intend for users to deal directly with the microcode of the calculators. The microcode implemented user-level functions, and on the programmable calculators, a user program consisted of a series of steps that invoked these functions. However, users were naturally curious about the internal workings of their calculators, and did quite a bit of reverse-engineering. Tom Napier published a series of articles describing his investigations into the inner workings of the HP-67, and a plotter interface he constructed[PPCV5N7P7]. However, the HP-67 was not a platform conducive to user microcode development, as there was a fixed amount of microcode built into the machine, with no provision for expansion.

The HP-41C, introduced in 1979, was HP's first expandable scientific handheld calculator. Its four expansion ports could hold RAM, ROM, or peripheral interfaces, the latter of which also contained ROMs. Significantly, the ROMs could contain user-level programs, microcode, or a combination of the two. HP published some details about the internal organization of the HP-41C in the PPC Journal “Corvallis Division Column”, such as the HP-41C Main Function Table[PPCV6N4P11], HP-41C Postfix Table[PPCV6N5P20], HP-41C Data and Program Structure[PPCV6N6P19]
Armed with this information many enthusiastic users set about using bugs in

the ROM to investigate unintended behavior of the calculator, bringing about "Synthetic Programming". The information eventually also proved invaluable for user microcode development.

Kelly McClellan, Jim De Arras and other users reverse-engineered the Nut processor bus and instruction set[PPCV6N6P4][PPCV7N3P20], and built EPROM boxes and MLDL (RAM) devices in order to run their own microcode.

HP eventually was persuaded to release the source code of the HP-41C ROMs and some of the accessory ROMs to PPC on a NOMAS (NOt MAnufacturer Supported) basis, further encouraging user microcode development.

Microcode-Level Simulation

Microcode-level simulation can avoid the problems of functional simulation, and microcode-level simulators are generally easier to develop than a complex functional simulator (e.g., HP-15C), because the microarchitecture is by design very simple in order to minimize the hardware cost. If the microcode-level simulator implements the microarchitecture correctly, and the actual microcode from the calculator is used, the functioning of the calculator can be reproduced nearly perfectly, including synthetic programming and bugs.

HP undoubtedly used microcode-level simulation during the development of the calculators in order to debug the microcode before hardware was available. HP even offered the HP 45423A Financial Calculator software for the HP 150 "Touchscreen" computer, which consisted of microcode-level simulations of the HP-11C scientific and HP-12C financial calculators.

The principle drawback of microcode-level simulation is performance. Because many more details must be accurately simulated, a microcode-level simulator almost always requires more host CPU performance than a functional simulator in order to provide a comparable simulation rate. In the case of early HP calculators, this is generally not an issue because the actual calculator hardware was extremely slow as compared to modern processors in desktop computers and even PDAs.

Most users did not write their own microcode, but instead purchased third-party ROMs containing microcoded functions, such as the ZENROM or AECROM.

Due to the common use of both synthetic programming and third-party microcode on the HP-41C, a microcode-level simulator is much more useful than a functional simulator.

The Nonpareil Project

The Nut instruction set and the HP-41C internals were well understood by the mid 1980s, and some microcode-level simulators had been written, but none were publicly available. In 1993, my curiousity about the internals of the earlier calculators drove me to research the U.S. Patents granted to HP regarding calculators. I found that there were some early patents that contained a detailed description of the instruction set of the first-generation calculators (HP-35, HP-80, etc.), along with the complete microcode source listings for the HP-45, HP-55, and HP-80 calculators. I typed in the listings, and wrote a microassembler and microcode-level simulator, which I made available as free software under the General Public License[GPLv2], by the unwieldy name “CASMSIM”.

Shortly thereafter, I wrote a Nut microcode-level simulator “NSIM”, which I also released under the GPL.

Since that time, others have published more sophisticated microcode-level simulators for the HP-41C and for Saturn-based calculators including the HP-48 family. Since those seem adequately represented, my own focus has remained on early HP calculators. After a hiatus of several years, I resumed work on microcode-level simulators in 2003. I started reverse-engineering the “Woodstock” processor used in the second generation calculators (e.g., HP-25, HP-67), and got the HP-25 code running. I was pleasantly surprised to find that the “Spice” series uses the same instruction set, so I was able to get the HP-33C running in simulation fairly quickly.

Much of the simulation code can be shared between calculator families, so rather than continue to maintain independent simulators, I merged them into a single simulator now named “Nonpareil” (without equal). Nonpareil also is the name of a particular type of confection, and an alternate name for a bird, the painted bunting, so I will probably adopt one of those for the Nonpareil logo.

ROM Dumping

A microcode-level simulation obviously requires microcode to be useful. HP generally only provided the microcode embodied in ROM chips, so it is necessary to somehow extract a ROM image or “dump”. Due to their expandability and powerful programming features, users quickly learned how to dump the ROMs of the HP-41C and HP-48. Some Saturn-based calculators such as the HP-17B, HP-27S, and HP-42S include a memory viewer that is capable of dumping memory to an HP 82440 infrared printer or an HP-48 or other device that can receive the IR printer protocol. For other calculators, ROM dumping generally requires opening the case and making electrical connections to the circuitry.

This form of ROM dumping is easiest on the calculators that have a built-in self-test function. This first appeared on the Spice series (e.g., HP-33E), and was activated by the key sequence STO ENTER^. The self-test verifies that the ROM contents are correct[2] and that the processor and RAM work correctly. As this self-test takes place, it is possible to passively monitor the internal bus of the calculator using a logic analyzer, and capture the contents of each ROM word, so (mostly) noninvasive ROM dump is possible.

The Voyager series and the Saturn-based calculators also have a similar self-test, usually invoked by pressing the ON key simultaneously with another key.

The Voyager calculators were the first ones from which I captured ROM dumps. Use of a typical logic analyzer to capture ROM contents is difficult unless the analyzer has very deep capture memory, because the bus is serial. I have built several models of dedicated hardware devices which I call "ROMsuckers" for this specific purpose. The second generation ROMsucker used a Linear Technology LT1045 hex level translator to convert the calculator signals (PMOS, NMOS, or CMOS, with voltage ranges from -12.5V to +12.5V) to 5V CMOS levels. These signals drive a DLP-USB245M module which interfaces to USB.

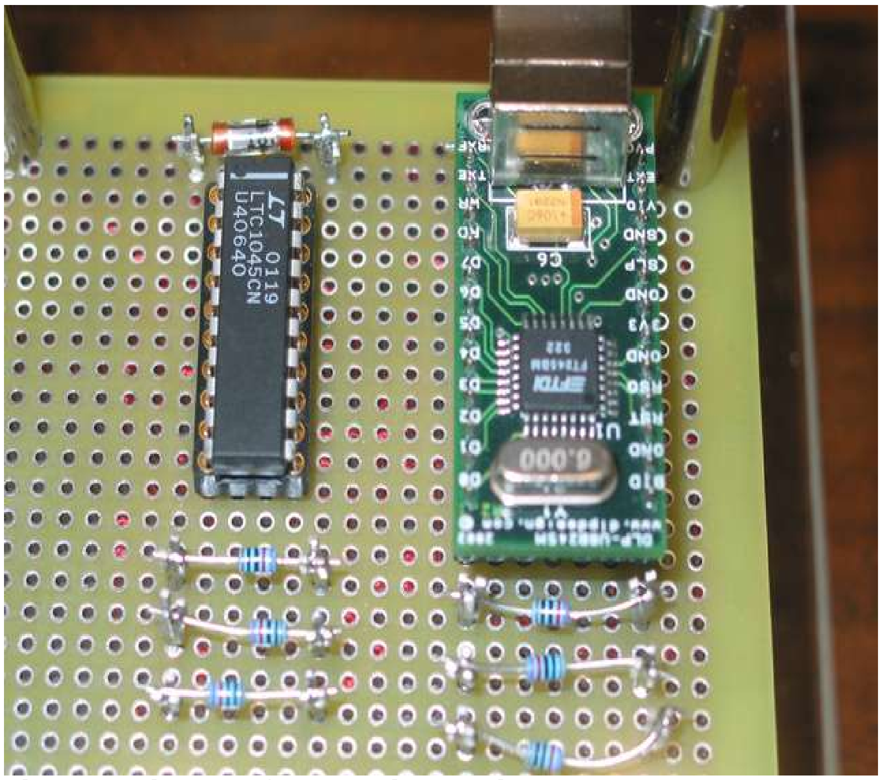


Photo 1: Closeup of ROMsucker Mark 2

2 All single-bit errors and most multi-bit errors will be detected.

A Linux program named “suck” was written to capture the data into a binary file, and a program “dumplog” decodes it into a (barely) human-readable text file.

With the ROMsucker Mark 2, I was able to dump the ROMs from the HP-11C, HP-12C, HP-15C, and HP-16C. I have not yet dumped the ROMs from an HP-10C. The HP-10C is fairly rare. I have only two units. One is physically pristine, and I am loathe to take it apart. The other is in rough condition, so I don't care so much about it. However, it uses the early Voyager style construction in which everything is wrapped in antistatic shielding, and the chips are on a display module separate from the keyboard. This is more challenging to probe, though I expect to get to it eventually.

In the Voyager calculators, the ROM checksum is computed using a software loop. The CXISA instruction (new to the Nut processor) is used to read each word.

In the earlier Spice calculators, I was very surprised to discover that there is no equivalent of the CXISA instruction. Instead, the Spice CPU has a single instruction which when executed takes 1024 bus cycles to read an entire 1K word block of ROM, and computes the checksum in hardware.

On the calculators prior to the Spice series, dumping the ROM is significantly more difficult. Without a self-test function, and only passive monitoring, one must exercise all the functions of the calculator in an attempt to force it to execute all the ROM instructions. I have experimented with this approach using a version of Nonpareil instrumented for code coverage analysis. Although I was able to find a sequence of key presses to read all of the actually used ROM words of the HP-45, the problem has proven to be more difficult on more sophisticated calculators. If you can't get all the ROM words dumped this way, it is not easy to determine whether the missing ROM words are unused, or that you simply haven't found the condition that invokes them.

I plan to build a future version of the ROMsucker that can operate in an “invasive” mode. This will require disconnecting the ISA signal that carries the instruciton address and data between the CPU and ROM chips. (In the first generation calculators, the instruction and address are on the separate IS and IA lines, and only IA would need to be disconnected.) This allows the ROMsucker to inject addresses directly. Due to the implementation of bank switching in the ROM chips (discussed in more detail below), this will require tricky firmware in the ROMsucker. HP designed the Topcat series of calculators with a jumper in the ISA line, apparently specifically for diagnostic purposes.

Some precaution will be necessary when using an invasive ROMsucker with printing calculators. PPC members who experimented with NNNs (non-normalized numbers) on the HP-97 discovered that it was possible for the calculator to get into states where heating elements of the printhead were turned on but not turned off again in the required interval, thus permanently damaging the printhead. It is possible that the intrusive ROMsucker could put the calculator into such a state. I believe that the calculator will work fine with the printhead flex circuit disconnected from the driver board.

An alternate approach was used by Peter Monta to dump the ROM code from an HP-35. In a ROM chip, the bits are encoded by the presence or absence of metalization at a particular location in the memory cell. In older semiconductor technology, there was only a single layer of ROM, which was the topmost layer (though possibly under a passivation coating), so it can be seen optically. Peter cut the tops from the metal-can ROM packages, photomicrographed the ROM chips, and wrote software to extract the ROM bits from the resulting images.

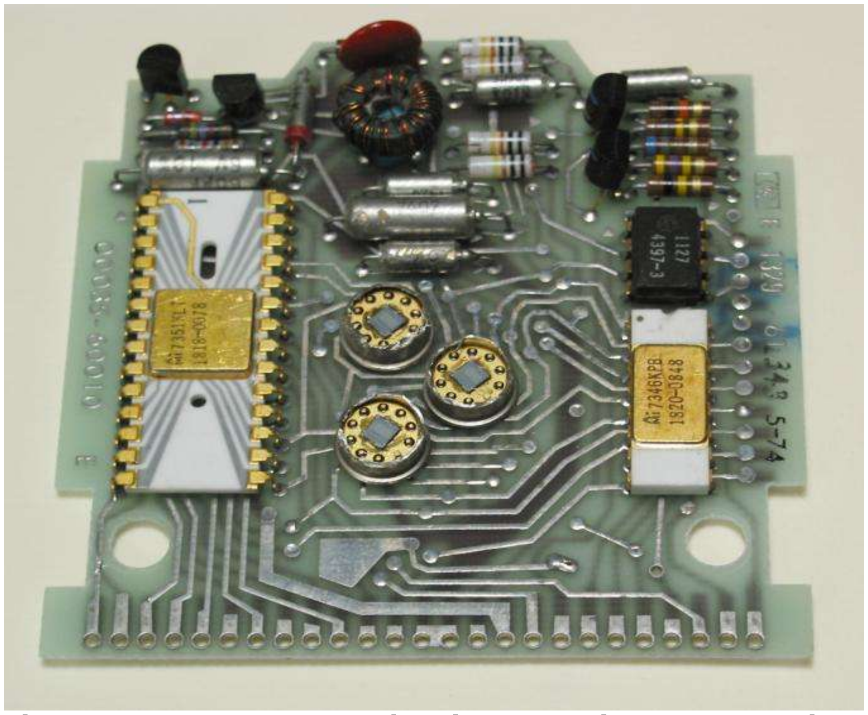


Photo 2: HP-35 Circuit Board With ROM Packages Decapped

Photo by Peter Monta

Peter had to spend some time studying the bits and the visible structure of the address decode array in the ROM chip to determine the correct mapping of physical location to logical address and bit position.
A description of the procedure and more photographs can be seen on Peter's

web page, at the URL listed in the Web References section at the end of this paper.

ROM extraction from some HP models will be particularly difficult. The ROMs of the HP-01 watch are inside a ceramic hybrid module. There do appear to be some test points on the module, but it is unclear whether any of them provide access to the necessary clock, sync, address, and instruction signals. If not, optical extraction may be the only viable option. If that proves necessary, I would much prefer to attempt it on a broken HP-01 rather than a working one. However, I do not have a broken HP-01, and even broken ones seem to fetch high prices on eBay.

Note that there is a patent on the HP-01 which contains a source listing of a prerelease of the ROM. However, this is definitely not final ROM code as the keyboard layout is subtly different than production units, and there are nine 256-word pages of ROM listed, while the HP Journal article on the HP-01 states that it contains two quad ROMs. Of the nine 256-word pages, there are over 200 unused locations, so it is quite likely that HP engineers were able to squeeze the code to fit in two quads. It should be possible to get this prerelease HP-01 code working on a simulator, and it would be quite desirable to do so before destroying an actual HP-01 to extract the final ROM code.

The HP-10 “KISS” basic printing calculator (not the later HP-10B or HP-10C LCD calculators) appears to have the CPU and ROM integrated into a single chip. Tony Duell has traced out the schematic and states that there are no extra pins that could provide the necessary bus signals. Optical extraction is probably the only viable method. As with the HP-01, the HP-10 is rare, so I'd rather do this to a broken one, but have not located one.

The more sophisticated Saturn-based calculators generally have a separate ROM which could relatively easily be dumped. However, most of the lower-end Saturn-based calculators have the CPU and ROM on a single chip. Internal HP engineering documents reveal that these integrated Saturn chips include a test mode that makes the internal Saturn bus cycles available on pins of the package. Unfortunately I have no information on the pinouts, and it will be a lot of work to determine how to invoke this test mode by trial and error.

ROM Bank Switching

The first-generation HP handheld calculators used ROMs storing 256 instruction words each. There was a cumbersome mechanism for selecting one of eight ROMs, allowing for a maximum of 2048 instructions if eight ROMs were used. The processor was not involved in ROM selection, so the program counter and the single-level return stack could only address the currently selected ROM.

Later ROMs in the first-generation handhelds stored 1024 words (one “quad”), but still functioned as the equivalent of four 256-word ROMs.

The HP-55 and HP-65 needed more than 2K instructions, so later ROM chips supported selection of one of two “groups”, allowing for up to 4K instructions. These models actually used 3K of ROM; no handhelds based on that chipset used more than 3K. These models also introduced the concept of a delayed ROM select, so that a ROM change would occur after a following branch instruction, simplifying the programmer's task in arranging the program.

The HP-46, HP-81, and HP-9805 desktop calculators used the same chipset with more ROM, but added external selection hardware.

In the second-generation chipset introduced with the Woodstock series of calculators (e.g., HP-25), the processor became more directly involved in ROM addressing. The program counter and two-level return stack were 12 bits wide, providing for a maximum of 4096 instruction words. Perhaps at the time it was designed this was deemed to offer some headroom, but in fact it was not long until it became insufficient. The HP-92, HP-95C, HP-67/97, and HP-19C all used more than 4K, so bank switching had to be introduced again. In the Spice series, the HP-38E, HP-38C, and HP-34C also use bank switching.

I have reverse-engineered the bank switching used in the Spice series. While I suspect that the Woodstock bank switching is similar or identical, I have not yet verified this.

My initial expectation was that the Spice bank switching would be very similar to that used in the HP-41CX and some late HP-41C ROM modules. Those have two instructions to select bank 1 and bank 2. Executing one of those instructions from a bank-switched 4K block of ROM affects the bank selection of that block of ROM only.

However, the Spice bank switching is in fact much simpler. There is only a single bank switching instruction, which acts as a toggle, and affects all bank-switched ROMs in the calculator regardless of address space.

This single-instruction bank toggle scheme has interesting consequences for the ROM self-test. During the self-test procedure, the processor reads all of the ROM words of a 1K block of ROM with the SYNC line activated for each word. This means that the ROM cannot distinguish a self-test fetch from a normal instruction fetch. Thus if a bank-switch instruction is fetched during the self-test, the following ROM words will be fetched from the opposite bank, until another bank-switch instruction is fetched. Thus the ROM checksums of the bank-switched blocks of the address space are not each computed over a single coherent bank, but rather on a mix of words for both banks.

Once I realized that this interaction might be occurring, I hypothesized that any Spice bank-switched address block would contain bank-switch instructions at the same addresses in both banks. Otherwise it would be impossible for the self-test procedure to verify all words of both banks. Furthermore, it seemed likely that there would be an even number of bank switch instructions in each bank, so that the self-test instruction would complete with the same bank selected as when it began. Examination of the dumps from the ROMsucker verified both hypotheses for all three bank-switched models.

Once I updated my “dumplog” program to account for the bank-switching, the HP-38E, HP-38C, and HP-34C code started partially working in the simulator.

Multiple ROM Versions

Some calculator models were produced with multiple ROM revisions, usually in order to correct bugs. The HP-35, HP-45, HP-67/97, and HP-41C were famous for this. In the case of the HP-45, in addition to fixing bugs, the operation of storage arithmetic was significantly changed.

Some models had firmware changes that were less well known outside HP. For instance, there apparently were bug fixes to the HP-80 and HP-11C.

Saturn-based calculators generally have a means of indicating the ROM revision on the display, but with earlier calculators (other than the HP-41C), one can only test for the presence or absence of known bugs, or use hardware means to dump the ROM contents for comparison.

Simulation Timing

The most obvious choices for timing for a simulator are to match the real time execution rate of the actual calculator, or to run at the maximum possible simulation speed. The latter is actually easier to implement, but can cause some difficulties. For instance, the calculator microcode often uses timing loops for button debouncing, so running at a faster than normal rate can prevent the keyboard from operating correctly. In the HP-41C, if a key is held for approximately one second, the function name in the display is replaced by “NULL”, and no operation is performed when the key is released. This feature is intended to make it easier to use user-defined keys, since you might forget what function you have assigned to a key. But if the simulation runs faster than normal, the NULL appears too quickly.

These timing problems with fast simulation can be solved by either modifying the calculator ROM image to take the higher speed into account, or by having the simulator detect the timing loop and take corrective action. Automatic detection of the necessary compensation points is not always easy, although a good first-order approximation would be to monitor all simulated instructions that reference the key pressed flag. Appropriate corrective action might be to adjust a loop counter in a simulated register, or to temporarily reduce the simulation speed.

Nonpareil currently attempts to use real-time execution speed. Prior to Saturn, HP handheld calculators executed microinstructions at a fixed rate. For instance, the HP-55 executes 3500 microinstructions per second (based on a crystal oscillator). The microcode for the HP-55 timer mode depends on this rate. Nonpareil matches it by timing the execution of a small number of instructions then suspending execution for the amount of time needed for a real calculator to “catch up”.

Simulator Program Structure

CSIM and NSIM were written as single processes, with display update and keyboard polling happening periodically. Nonpareil instead uses two processes, with one handling user interface and one running the simulation core. Semaphores and mutexes are used for synchronization between the two processes.

The two-process structure is also helpful in implementing debugging features, as the GUI process can easily set breakpoints and control the execution of the simulator process. This is useful for reverse-engineering existing microcode, debugging newly written microcode, or debugging the simulation core itself.

User Interface

CSIM and NSIM were written with the GUI code segregated into a single source file that could be easily rewritten for alternate operating systems. The standard releases originally used Xlib (X Window System) graphics primitives, though I later switched to the GTK+ toolkit. GTK+ has the advantage that it is available on multiple platforms, although only the Linux/Unix platform seems mature at this time. Nonpareil also uses glib, a portability library associated with GTK+, to provide platform-independent timing and process synchronization primitives.

CSIM and NSIM originally used graphics primitives to render a very crude approximation of the appearance of the calculator, as can be seen in figures 1 and 2.

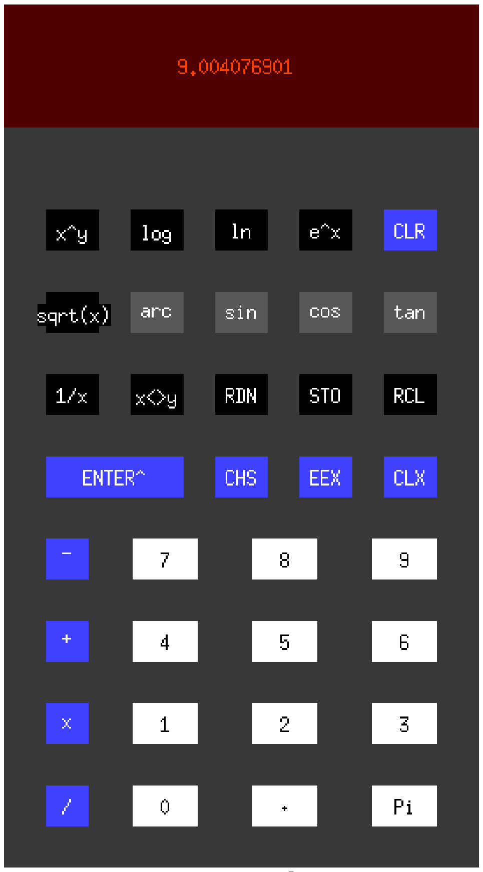


Figure 1: CSIM simulating HP-35

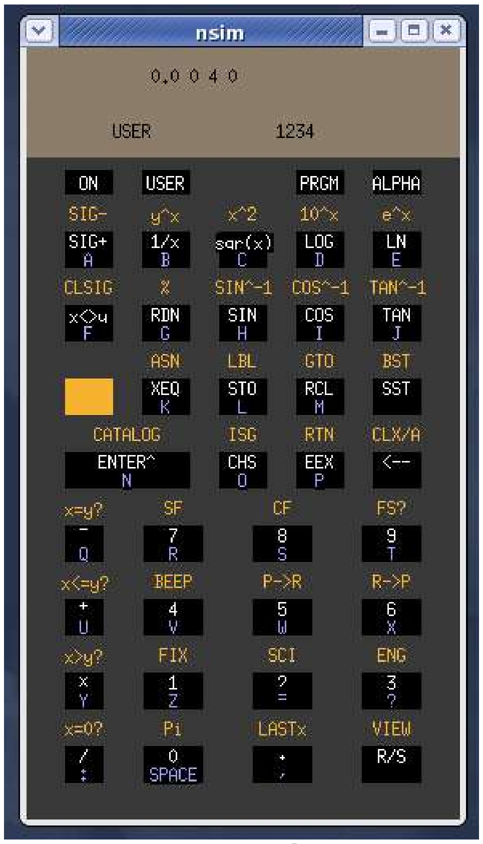
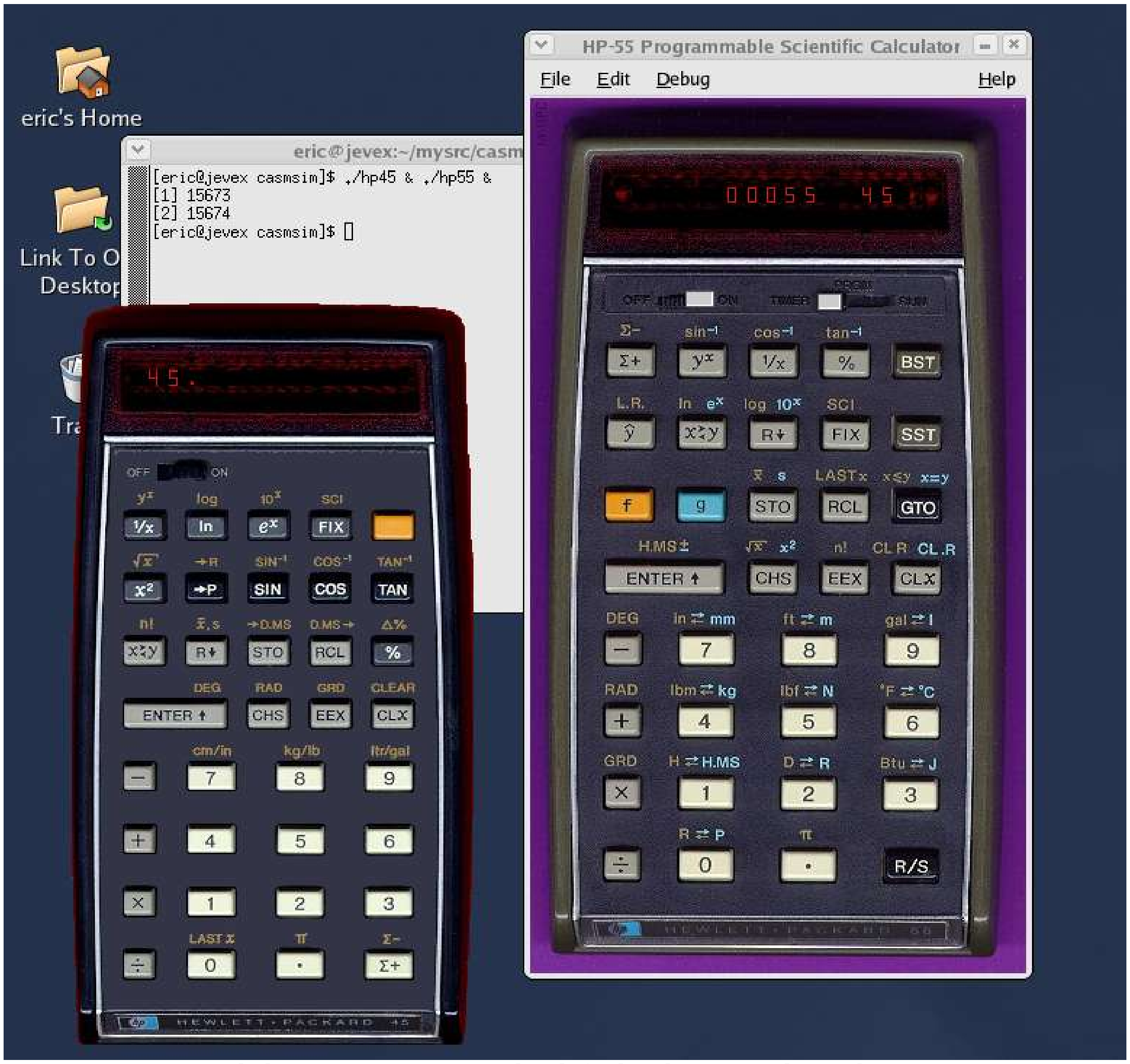


Figure 2: NSIM simulating HP-41CV

Nonpareil instead loads a PNG file to serve as the background of the calculator window. The PNG file can be a photograph or scan of an actual calculator, and David Hicks of the Museum of HP Calculators has graciously given me permission to use some photographs from his web site in the Nonpareil package.

By default, Nonpareil creates the calculator window with no menu bar or other adornments, and adjusts the shape of the windows based on transparency in the PNG file, so that an HP-35 window is shaped like the outline of an HP-35, as can bee seen in figure 3. This “shape” mode can be disabled to provide a normal window with a menu bar, close box, etc.

In the default “shape” mode, the menus will pop up if you click the right mouse button on the calculator display area, and the window can be moved around the desktop by clicking and dragging with the left mouse button.



```
[eric@jevex casmsim]$ ./hp45 & ./hp55 &
[1] 15673
[2] 15674
[eric@jevex casmsim]$ []
```

Figure 3: Nonpareil showing an HP-45 in the default "shape" mode, and an HP-55 in "non-shape" mode

KML Language for Simulator Configuration

The Nonpareil user interface is defined by a KML file. Each calculator to be simulated has one or more KML files, and the user specifies one on the command line. The KML file tells Nonpareil what hardware to simulate, where to find the ROM and graphics files, and the locations and sizes of the buttons.

KML was defined by Sébastien Carlier and Casey Patterson for use in Emu48, and the specification is currently maintained by Christoph Gießelink.

Since the KML specification originally only had support for bitmapped displays, I have added several extensions to KML to support segment-oriented displays as used on the calculators with LED displays and seven-segment and fourteen-segment LCD displays. Other KML extensions have also been added. It would probably be a good idea to try to merge the Nonpareil changes into the official KML specification, although I am also considering switching from KML to XML.

Legal Issues

I am not a lawyer, and do not offer any specific legal advice. However, there are some potential legal issues that may face developers of simulators (functional or microcode-level):

- Copyright:

It appears that there is no U.S. Copyright on the firmware of any HP handheld calculator introduced before 1983. Prior to the US adoption of the Berne Convention, a work was required to bear a copyright notice in order to receive copyright protection. There were certain very limited conditions where the accidental omission of the copyright notice could be corrected, but it does not appear that HP calculators qualify. Details may be found in 17 U.S.C. 405(a).

The HP-71B handheld computers and all HP handheld calculators introduced after 1983 have copyright notices, so it is generally not possible to distribute the ROM images without explicit permission from HP. HP has granted permission for ROM images of some specific models to be publicly distributed.

- Patents:

The U.S. patents on early HP calculators have expired. There are still U.S. patents in force regarding aspects of RPL, fraction entry and display, the equation editor, etc.

- Trademarks:

There are potential trademark problems with use of the name “Hewlett-Packard” or the initials “HP” in the name, description, or advertising of a simulator.

Current Status

Classic series: HP-35, HP-45, HP-55, HP-80 believed working correctly.
- Woodstock series: HP-25 partially working

- Spice series: HP-32E believed working correctly. Other models partially work.
- Coconut: HP-41CV working, but state save not yet implemented, and fourteen-segment LCD display needs improvement.
- Voyager series: Displays memory lost, then a zero X register, but no further useful behavior. Seven-segment LCD display needs improvement.

Ports

There are not yet any ports of Nonpareil to other platforms, but there are several ports of my earlier simulators:

David Hicks of the Museum of HP Calculators has ported it as a Java applet:
 http://www.hpmuseum.org/simulate/sim45.htm

Jonathan Purvis has ported CSIM to PalmOS:
 http://one-two-three-four-five.com/palm/csim/

Maciej Bartosiak ported NSIM to MacOS X and significantly improved the appearance:
 http://homepage.mac.com/mba/nsim/
Maciej has graciously allowed me to incorporate some of his graphical improvements into Nonpareil, and tells me that he is working on porting Nonpareil to MacOS X.

Custom Hardware

Since certain models of old HP calculators are now difficult to obtain and/or expensive, there may be some interest in building replicas.

Richard Ottosen has built a custom calculator hardware prototype named "DIY-RPN" using a PIC18F452 microcontroller and a 16*2 character LCD module. I have written Woodstock simulation code in assembly language to run on it, and am in the process of debugging it. The component side of the printed circuit board is shown in photo 3.

The DIY-RPN keyboard uses an array of surface-mount "tact switches", which use metal snap-disc similar to (but smaller than) those of traditional HP keyboards. These can be seen in photo 4. Of course, we do not have custom double-shot injection-molded keys. Instead, we are currently using an overlay. Richard purchased a photo printer and scaled and modified a photograph of an HP-25. It works surprisingly well.

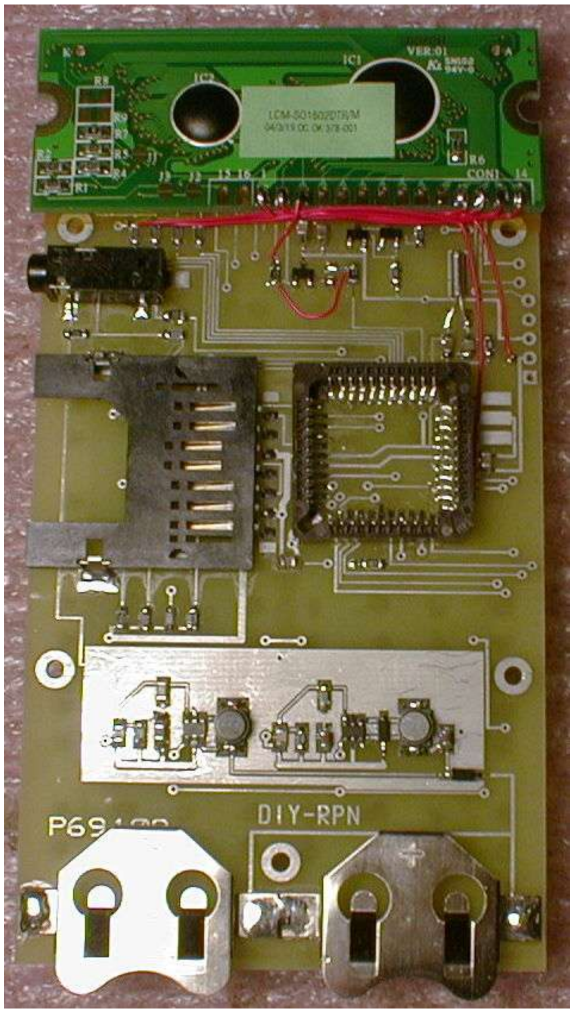


Photo 3: DIY-RPN component side

Microcontroller needs to be installed in square socket. LCD module at top, serial and SD card jacks on left, battery holders at bottom. Two boost switching regulators surrounded by a rectangular ground plane provide regulated 5V for the microcontroller and LCD module when the calculator is turned on, and regulated 3.3V for the SD card when it is active.

Photo by Richard Ottosen.

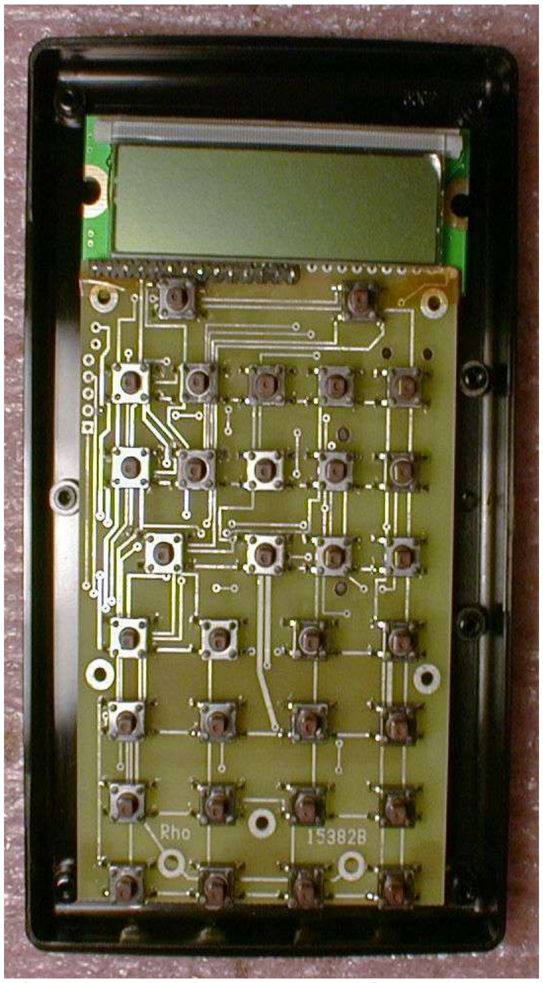


Photo 4: DIY-RPN circuit side

Surface mount tact switches and the front of the LCD module. When assembled, the buttons protrude through holes in the top case, and a photo paper overlay provides keyboard legends.

Photo by Richard Ottosen.

Klaus (last name unknown) has built calculator hardware using a PIC16F877 microcontroller. His software is written in C, using a portion of the Nonpareil code. The schematics and source code (GPL) are available from his web site, listed below in the Web References section.

Future Work

Arguably the highest priority task is to find and fix bugs in the processor simulation. The bug affecting the logarithmic and exponential functions of the HP-25 is particularly exasperating. On several occasions I have found errors in the code which seemed likely to be the cause of the log/exp bug, only to find that the correction has not in fact eliminated the bug.

I expect that fixing the underlying bug causing the log/exp problem may also fix the problems with various Spice series models.

I have captured traces from several Spice runs for comparison with simulator traces, to attempt to find the differences. Unfortunately the differences tend to occur on conditional branches after hundreds or thousands of instruction cycles. If the simulator takes the wrong path of the conditional branch, it is not usually immediately obvious why this occurs, because the traces of the real hardware don't provide direct visibility of internal register state.

There are still many calculator models for which more reverse-engineering will be necessary. Currently I have no information on the operation of the card readers in the HP-65 and HP-67/97, and only a brief description of the printer instructions used in the HP-91, HP-92, HP-95C, and HP-97.

I have started work on a Windows port of Nonpareil. The simulation core seems to work, but bugs in the Windows port of the GTK+ toolkit result in missing display updates. There is also a bug in the GTK+ support for unadorned windows, resulting in the window shape mask being applied at an offset.

Maciej Bartosiak has stated that he is working on a MacOS X port of Nonpareil.

Web References

Nonpareil: http://nonpareil.brouhaha.com/
CASMSIM: http://www.brouhaha.com/~eric/software/casmsim/
NSIM: http://www.brouhaha.com/~eric/software/nsim/

Photos of the ROMsucker:
http://gallery.brouhaha.com/romsucker

Museum of HP Calculators:
http://www.hpmuseum.org/

Peter Monta's "Reverse Engineering the HP-35":
http://www.pmonta.com/calculators/hp-35/

Klaus's IQSIM:
http://www.informatik.fh-wiesbaden.de/~khind001/iqsim/iqsim.html

Christoph Gießelink's EMU48 page, including the KML specification:
http://privat.swol.de/ChristophGiesselink/emu48.htm

Acknowledgments

The author gratefully acknowledges assistance and information provided by many individuals:

- Maciej Bartosiak
- Luiz Cláudio
- Tony Duell
- Bob Edelen
- Christoph Gießelink
- Dave Hicks
- Wieland Hingst
- Wlodek Mier-Jedrzejowicz
- Peter Monta
- Richard Ottosen
- Nelson Sicuro
- Mark Sims
- Randy Sloyer
- Ken Sumrall

My apologies if I've omitted anyone.

Bibliography

Wickes1980: , Synthetic Programming on the HP-41C, 1980
Wilkes1951: M.V. Wilkes, The Best Way to Design an Automated Calculating Machine, 1951
Cochran1968: David S. Cochran, Internal Programming of the 9100A Calculator, 1968
Osborne1982: Thomas E. Osborne, Hewlett-Packard Calculator Architectures, 1982
Dickie84: James P. Dickie, Custom CMOS Architecture for a Handheld Computer, 1984