HEWLETT-PACKARD JOURNAL

June 1991 Volume 42 • Number 3

Articles

The HP 48SX Scientific Expandable Calculator: Innovation and Evolution

Many of the features of this advanced handheld calculator have evolved from its predecessors, the HP 41C and HP 28S. Others, such as its unit management system, are new.

by William C. Wickes and Charles M. Patton

SINCE THE INTRODUCTION OF THE HP 65 in 1974, Hewlett-Packard has developed a succession of customizable scientific calculators of ever expanding capability. The HP 48SX scientific expandable calculator (Fig. 1) maintains this trend with an unprecedented combination of features and flexibility. Its major features include:

- An RPN-style calculator interface with a dynamic stack of arbitrary depth for operations on eighteen types of mathematical or logical objects (plus twelve additional object types used by system programs).
- Numerous arithmetic, transcendental, and statistical functions, applied uniformly wherever meaningful to complex as well as real numbers.
- Vector and matrix operations on real and complex arrays of arbitrary size. A spreadsheet-like screen editor is provided for simplified entry and editing of arrays.
- Symbolic mathematics including evaluation, expansion, simplification, summation, differentiation, and integration. The EquationWriter application provides graphical, "textbook" entry of expressions and equations.
- String manipulations.
- Binary integer operations, with arithmetic, bit, and byte manipulations, and a variable word size.
- Numerical and symbolic equation solving.
- Eight types of automatic mathematical and statistical plotting, including interactive root finding, calculus, labeling, and digitizing. There are also interactive and programmatic line and arc drawing and creation of custom text and graphics displays.
- An integrated unit management system. Quantities that include physical units can be used in computations, solving, and plotting, while the calculator automatically performs unit conversions and dimension checking.
- Time management, including a clock/date display and appointment and program-execution alarms.
- Two-way communications via a wired serial port for connection to personal computers, printers, and other serial devices or via infrared light for printing to the HP 82240A/B printer or for wireless transfers between two HP 48SXs.
- Customization with plug-in 32K-byte or 128K-byte RAM or ROM memory cards, which may include command libraries for extending the built-in feature set. Libraries can also be imported into RAM using either I/O mechanism.
- A user-definable keyboard and custom menus.
- Programming in the RPL language, which provides program-flow structures, recursion, global and local variables, passing procedures as arguments, input prompting and output labeling, user and system flags, logical tests and operations, and user-defined functions.

Fig. 1. *HP 48SX scientific expandable calculator, showing the EquationWriter application.*

These features are supported by a hardware set that includes a vertical-format package with 49 keys, a 131-by-64-pixel LCD display with support for fast scrolling of virtual displays that are larger than the physical screen, two plug-in slots for memory cards, a four-wire serial communications port, and an infrared transmitter and receiver (see articles, page 25 and 35).

Design Objectives

The fundamental design objective for the HP 48SX was to create a product that combines the software technology of the HP 28S[1] with the hardware flexibility and customizability of the HP 41C.[2] Although in many respects the HP 28S was itself a descendant of the HP 41C, its advanced capabilities and limited hardware have made its application and range of customers somewhat different from those of the HP 41C. For example, in the academic field, the HP 41C was very popular in college engineering departments, but it had little appeal to mathematics instructors. By contrast, the HP 28S has had a significant effect on mathematics instruction, with many colleges adopting it as a standard teaching tool. Engineering departments have been much slower to adopt the HP 28S, since it does not have the software exchange capabilities they are accustomed to with the HP 41C. Similarly, the HP 41C was very popular with surveyors, but the HP 28S is of limited use in this field because of its lack of I/O capability.

The HP 48SX project started, therefore, with a review of the strengths of its two predecessors. The HP 41C's include plug-in memory ports, HP-IL I/O capability, a redefinable keyboard, and a vertical format convenient for handheld operation. The HP 28S's include extensive real and symbolic mathematical capabilities, RPL operating system and user language, a graphics display, and a menu key system.

At the same time, we focused on common enhancement requests from HP 28S owners. These include a bigger display, more graphics and plotting features, I/O capability, especially for importing or saving software, symbolic integration, and more help from the calculator in using some of its more complicated features.

All of these strengths and enhancements are incorporated in the HP 48SX. In some cases, the implementation of one of these items evolved into a major feature that wasn't necessarily anticipated from the HP 28S/HP 41C combination or a customer request. For example, the HP 28S's powerful numerical integrator was obscured by an arcane syntax for entering the integration arguments. Consideration of this problem in the HP 48SX investigation led to a review of the general problem of entering and recognizing mathematical expressions, which ultimately led to the development of the EquationWriter application (see article, page 13). This solves the integration problem—one enters an integral by "drawing" a textbook-like expression on the screen, including the integral sign, upper and lower limits, and integrand, all appropriately positioned. However, the scope and utility of the EquationWriter far exceed what is needed for this particular use.

The HP 48SX also contains important features that derive more from "next bench" research than from HP 41C or the HP 28S strengths or from customer input. The prime example of these is the HP 48SX's unit management. Simple one-to-one physical unit conversions have been available on calculators for years. Several HP 41C plug-in modules improved on this by providing a general-purpose conversion mechanism which could calculate any conversion factor from input and output units specified as text strings. The *HP 41C Petroleum Fluids Pac* incorporates this mechanism into its calculations so that the user can include units for the values entered for the programs, and ask for answers in particular units. The HP 48SX takes advantage of its multiple-object-type operating system and symbolic manipulations to provide a new level of unit management, in which numerical quantities can have physical units attached to them and carried throughout arbitrary calculations. The collection and cancellation of units and conversions between dimensionally consistent different units are handled automatically by the calculator. For example, a problem such as, "How fast is an object traveling after accelerating at 1 m/s^2 for half a minute, if its initial speed was 20 mph?" reduces to

```
(1_m/s^2)*.5_min+20_mph EVAL
```

on the HP 48SX, which returns 871.1_mph. In HP 48SX notation, the underscore _ acts as an object type identifier linking a floating-point number with a unit expression which can contain arbitrary products, powers, and quotients of physical units. The HP 48SX has 121 units built into ROM, from which the user can construct arbitrary compound units. Unit objects are supported in numerical and symbolic calculations, plotting, equation solving, and integration. This HP 48SX capability removes a great deal of the drudgery from calculations involving physical units.

In one aspect of the HP 48SX design it was not possible to satisfy both HP 41C and HP 28S owners: programming language. To support its other design objectives, the HP 48SX needed to use an RPL operating system and language similar to that used in the HP 28S. Unfortunately, this meant that the considerable body of programs written for the HP 41C would not be executable directly on the HP 48SX. To solve this problem, the plug-in HP 82210A HP 41C emulator card provides a keyboard emulation of the HP 41C and the ability to execute HP 41C programs. The infrared port and the HP 82242A infrared printer module for the HP 41C can be used to transmit programs from the HP 41C to the HP 48SX for execution with the emulator card.

HP 28S users have a smaller problem in program conversion. The great majority of HP 28S commands can be executed without modification on the HP 48SX. Only a few commands are different, primarily those associated with display operations (and the integral command, as mentioned previously), and the various system flags have changed. With optional software, the HP 28S can also use its infrared printer output to "print" its programs to the HP 48SX, where they can be executed after minor or no modification.

Internal Mechanisms

The remainder of this article will discuss some of the mechanisms the HP 48SX uses to support its feature set. The memory maps shown in Figs. 2 through 7 illustrate the concepts discussed in this article. The implementations of many of the higher-level applications are discussed in the article on page 13.

The fundamental basis of the HP 48SX system is the RPL operating system, which occupies about 18K bytes of the system ROM. This system first appeared in the HP 18C Business Consultant calculator in 1986.[3] In brief, the system combines elements of Forth and Lisp, providing a multi-

Fig. 2. *Structure of a ROMPART.*

object RPN stack and direct and indirect threaded execution, with both atomic and composite objects, temporary (lambda) variables, and the ability to pass unevaluated procedures as arguments. The objects are similar to Forth words, containing the address of the executable code that defines the object and the data that makes up the body of the object. The object types provided in the initial versions of RPL were:

- Identifier class: identifiers (global names), temporary identifiers (local names), and ROM pointers (XLib names). These objects are used for storing and retrieving other objects.
- Procedure class: secondary (program) and code objects. These objects are executable.
- Data class: floating-point real and complex numbers, character and string objects, hexadecimal strings (binary integers), real and complex arrays, linked arrays, extended precision real and complex numbers, lists, symbolic (algebraic), unsigned short integers, library, RAM/ROM pair (directory). Under normal execution, these objects merely return themselves, as passive data. However, symbolic objects and lists are composite objects, and can be evaluated like procedure class objects.

The body of a composite object is a sequence of other objects terminated by an end marker that serves as a program return if the body is executed as a procedure.

To support HP 48SX operations, several additional data-class objects were added to the above list:

- Graphics object. These are LCD bit maps, used for storing and manipulating graphical images.
- Tagged object. A tagged object contains a text string plus another object. The text is used to label the object. Operations applied to the tagged object ignore the tag and apply themselves directly to the "inner" object. Thus, a program might return the tagged object Speed:10_m/s, where Speed is the tag. Executing 10 • (times) then returns 100_m/s.
- Unit object. This consists of a floating-point real number combined with an algebraic expression representing physical units.
- Backup object. This object is designed for the archival storage of a single object in an independently configured RAM port. The backup object contains a second object plus a name, a length field, and a checksum. The HP 48SX contains commands for storing and retrieving objects from within backup objects when the latter are installed in RAM ports.
- Library data object. This object provides a memory buffer for use by plug-in applications that need to preserve data between executions.

In addition to the new object types, three object types that were present in the HP 28S are given more visibility in the HP 48SX:

- In the HP 28S, a user can create a directory object stored in a variable, but has no access to the directory as an object. In the HP 48SX, a directory has the same status as other objects—it can be recalled to the stack, edited, copied, stored, and so on.
- Built-in commands in the HP 28S and HP 48SX are organized in libraries, which are similar to compiled directories in which the linked list of named objects is compiled to a table-driven organization. Name resolution of the objects within libraries is necessary during parsing, where text names are replaced by ROM pointers. The latter contain indexes into library object tables, which in turn provide for fast location of an object's name and executable code. In the HP 48SX, libraries are available as ordinary objects, so that a user can move libraries in and out of the calculator via one of the I/O ports or on plug-in memory cards. When a library is installed in HP 48SX memory, it extends the HP 48SX's language by adding its own internal commands to the built-in set.
- ROM pointers are visible to the HP 48SX user as XLib name objects, the library analog of the global names that provide access to objects stored in global variables in RAM. Executing an XLib name executes the object within a library that is associated with the name. XLib names

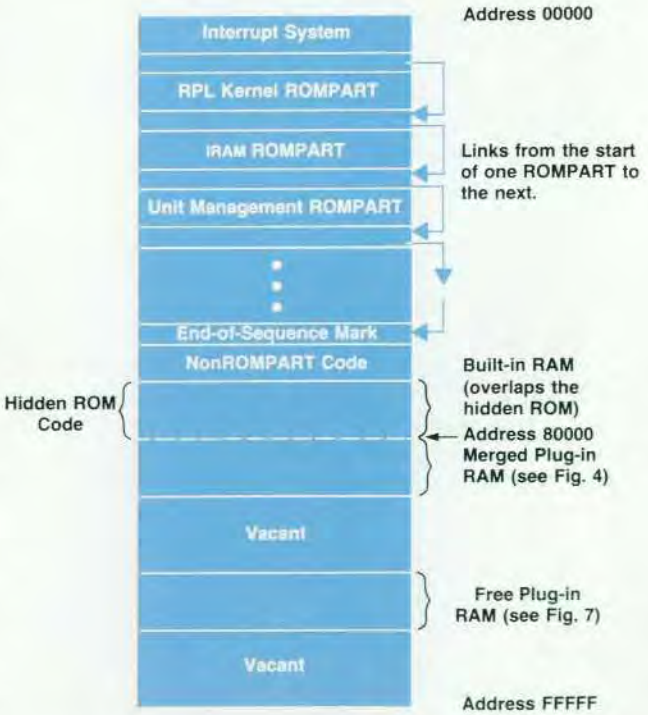


Fig. 3. *Overview of the address space layout of the HP 48SX calculator with one merged and one free (unmerged) RAM card.*

can be compiled as stand-alone RPN objects or included within the definitions of algebraic objects. As long as the relevant library is present, an XLib name decompiles to the text name stored in the library. If the library is removed, the XLib name is decompiled to show the library number and command number within the library.

As part of the strategy to maximize the use of ROM over RAM, the RPL system has included, from its inception, ROM-like structures that are analogs of the user's program and variable space (directories). These are called ROMPARTs. By attaching ROMPARTs to subdirectories, the user can create a context-sensitive customization so that typing the same thing can have very different results depending on the context directory. ROMPARTs were designed to provide context-sensitive customization, localizations of keywords and messages, system extensions, and run-time linking.

In the process of developing a set of programs, the user first creates programs and utilities in a customizing directory. When the programs are debugged and ready to keep, they can be transferred to a ROMPART and loaded into ROM. Attaching the ROMPART to the same directory provides the same functionality with less RAM use.

RPL Plug-in Management

While the overall scope and function of plug-in management did not change from the original RPL definition to its first released implementation in the HP 48SX, a number of design details did change in response to outside reviews of the system. Implementing these changes posed a number of design challenges.

ROMPART Structure. A large portion of the RPL plug-in management design is based on the concept of a ROMPART, which is not a standard RPL object in the same sense as complex numbers, directories, programs, and so on, but is instead a set of data-structure conventions.

A ROMPART is a collection of RPL objects together with a field containing the name of the ROMPART, a ROMPART ID number which uniquely identifies the ROMPART, a pointer to a hash table for use in identifying objects by name, a pointer to a link table for use in identifying objects by their unique object numbers, a pointer to a message table containing messages specific to this ROMPART, and a pointer to configuration code which is executed whenever the ROMPART needs to be configured (see Fig. 2).

The name field can contain any characters and is used only as information for the user. The ROMPART ID number uniquely identifies the ROMPART to the system and is in the range 000-7FE (hexadecimal). ID numbers 000-0FF are reserved for the RPL kernel and other built-in ROMPARTs. ID numbers 700-7FE are reserved for use by ROMPARTS providing application language extensions.

The hash table provides a two-way correspondence between names and objects in the ROMPART. It is used during the process of interpreting the characters typed in by the user to determine whether the characters name any object in the ROMPART, and then again to determine if an object should be displayed as a name rather than according to its internal structure.

The link table provides a list (in object-number order) of all accessible objects within the ROMPART.

Configuration. A ROMPART simply residing somewhere within the system does not provide for any of the services described above. The ROMPART needs to be registered with the system during the configuration process, which occurs in several stages.

Whenever the machine is first turned on, a routine check is made to see if any cards have been plugged in or removed from the system and adjustments are made to compensate for the changes. Then a number of known areas are scanned for the presence of ROMPARTs and a list of currently extant ROMPARTs and their locations is made and compared with the previous list. If no change is detected, the system continues the normal process of turning on the display and resuming the state it had when it was turned off.

On the other hand, if a change is detected, the system performs its warm-start code. Among other things, the warm-start code resets any pointers that could be referencing ROMPARTs that are no longer present. This includes the data and return stacks, updateable system pointers, and the ROMPART pointers connected to the home directory. The system then rebuilds the table of ROMPARTs and their

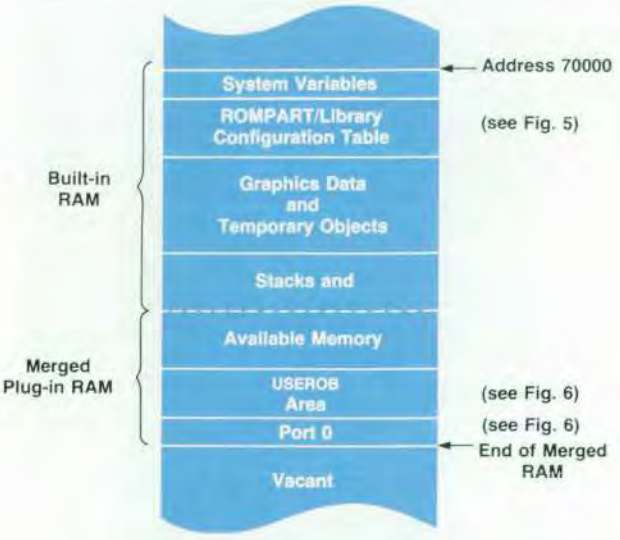


Fig. 4. *Overview of the layout of HP 48SX main RAM with one merged RAM card.*

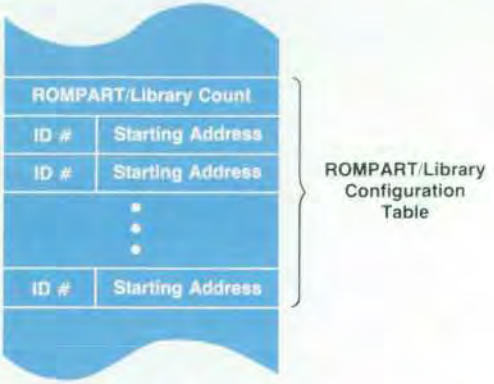


Fig. 5. *The ROMPART/library configuration table. Whenever the HP 48SX turns on, all ROMPARTs and libraries in ROM, port 0, and other unmerged plug-ins are registered here.*

locations and restarts RPL execution.

ROM Poll. One of the first things done when RPL execution is restarted is to proceed through the current list of ROMPARTs, executing each ROMPART's configuration code in turn. At this point, the RPL system is in a valid, stable state and the full resources of the system are available for use by the configuration code.

Although a ROMPART can take over the system at this point (as is done, for example, by the HP 48SX demo ROM), typical tasks are much less ambitious. More typical examples of tasks done at the ROM poll include:

- Attaching a ROMPART to the home directory so that the names of objects within the ROMPART are universally recognized.
- Replacing the hash and/or message tables of other ROMPARTs with versions localized for a particular language.
- Creating a custom subdirectory structure for use with this ROMPART.

ROM Pointers. The RPL system's ROM pointer (ROMPTR) objects provide a name and location independent method of specifying an object within a ROMPART. The data in a ROMPTR gives both the ROM ID number unique to a ROMPART and the particular object's object number.

As long as a ROMPART has been registered as being present in the system, and independent of whether it is attached to any directory, ROMPTRs referring to a ROMPART can be converted to the object or objects they specify. In this process, the current address of the ROMPART whose ROM ID is specified in the ROMPTR is found in the ROMPART table, and the ROMPART's link table is found. The object number specified by the ROMPTR is used as an index into this table to find the actual current address of the specified object.

ROMPTRs occupy a middle ground in terms of execution speed and flexibility between address pointers, which require no resolution but must be updated whenever memory moves, and ordinary identifiers, whose value can change in the course of execution but must be resolved by searching through the current context. Every programmable function and operation in the HP 48SX has an associated ROMPTR that specifies it. However, these are not normally used in programs since the address pointers will suffice. There is one case in which these ROMPTRs must be used, however. If the user stores a programmable function in a variable, what is stored is actually the corresponding ROMPTR, since storing an address pointer is contrary to the RPL conventions, and storing a copy of the object is clearly not what is desired.

ROMPTRs are normally created in the process of converting typed-in text to RPL objects (parsing). If the currently considered piece of text is not an object delimiter, number, or other fixed-syntax item, it is considered to be a name whose meaning is determined by the current context.

Names, ROMPTRs, and Localization. To determine the current interpretation of a name, the system first searches through all the variables in the current directory. If the name matches any one of these, the name is determined to be a variable name (ordinary identifier). If not, the system searches through the hash tables of the ROMPARTs attached to the current directory, if there are any. If a match is made, the name is determined to be a ROM word name, and it is converted to the corresponding ROMPTR. If no match is made, the search is continued with the parent directory, and so on. If the home directory is searched without a match, the name is determined to be a formal variable (also an ordinary identifier).

Every directory except the home directory can have at most one ROMPART attached to it. The hash table used in searching such a ROMPART is the one supplied with the ROMPART. The home directory, on the other hand, can have multiple ROMPARTs attached to it. Recorded with any ROMPART attached to the home directory is a pointer to its hash table. This allows the hash table provided by a ROMPART to be superseded by another hash table either in RAM or in another ROM (localization). Only ROMPARTs attached to the home directory can be so localized.

Structure of Plug-in Modules. The original RPL design provided for two kinds of plug-in modules: one associated with read-only devices (ROM) and one associated with read/write devices (RAM). Whenever a ROM device was detected, it was assumed that its data consisted of a linked list of ROMPARTs. The devices would be configured at some convenient but otherwise arbitrary address and the individual ROMPARTs would be registered as described previously. Whenever a new RAM device was detected, it was assumed that the device contained no viable data and the device would be configured to be a contiguous segment of the system's overall RAM, with current RAM contents

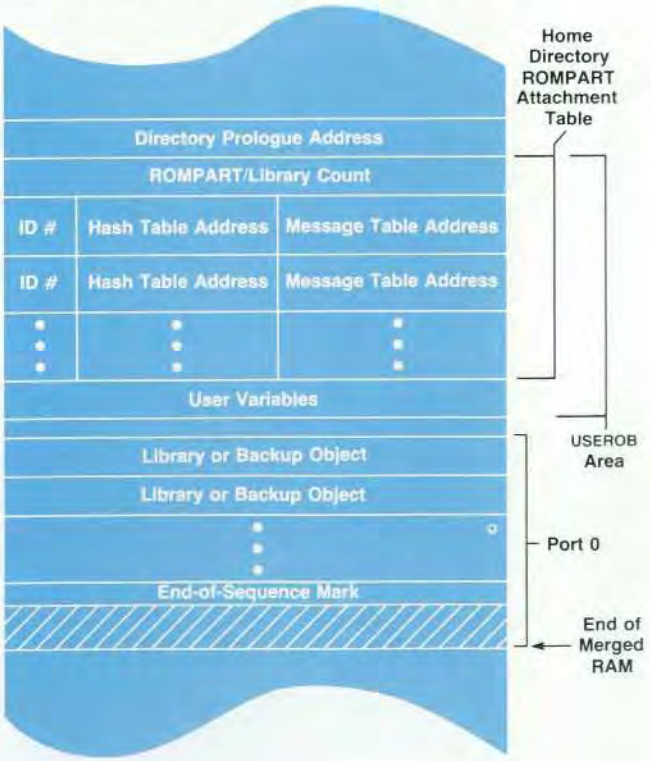


Fig. 6. *Layout of the* USEROB *area and port 0. A library or ROMPART attached to the home directory ROMPART attachment table, either by the* ATTACH *command or during the ROM poll, will have its keywords recognized by the system and can have both its keywords and its messages localized (for example, translated into other languages).*

shifted to new positions as necessary. This process is known as merging RAM. After RAM has been merged, it is not possible to unplug the module without endangering the system integrity. When a merged RAM is pulled, a large area of the system and user variables could go with it.

To make it possible to have ROMPARTs without read-only devices, an area within system RAM is set aside and given the same structure as a plug-in ROM, that is, a sequence of ROMPARTs. This RAM-based ROMPART area is also searched during the configuration of ROMPARTs. This model for plug-in structure provides almost all available services automatically and requires only five user-level commands: to prepare the system for removal of a RAM module (FREE) by reversing the procedure used to merge the RAM module, to attach and detach a given library from a given directory, to include a ROMPART (given in some object-coded form) in the RAM-based ROMPART area, and to remove such a ROMPART.

Design Changes and Challenges

In the evolution of the HP 48SX design, it became apparent that RAM modules needed to be used as mass-storage devices as well as system RAM. By analogy to flexible disks, one would expect that such a mass-storage RAM module could be removed without first informing the system. These two tenets had significant impact on the HP 48SX plug-in management design. Other factors that affected the design were that the RAM modules have a switch that allows them to act as ROM, that there is no effective way to determine the size of a ROM module, and that the system cannot reliably detect the removal of a plug-in as it is happening.

Backup Objects and Libraries. To use a RAM card as mass storage, we need to be able to store name/object pairs in the RAM card much as they are stored in variables in the main RAM. In addition, we need to be able to verify that the data on the card has not been corrupted in some way. This verification stage must be fast because it must happen at configuration time, that is, between the time the machine is turned on and the time the machine is available for use. Since no stand-alone object consisting of a name/object pair existed in the original RPL system, a new object type, the backup object, was invented for the purpose. A backup object, in addition to its prologue and length, consists of a name, an object, and a checksum.

Since ROMPARTs can coexist with backup objects in a plug-in, they are also encapsulated with a prologue and a checksum to become library objects.

The organization of the data in a ROM plug-in is largely dictated by the fact that the system can only determine the beginning of a ROM and not the end. This means that any data structure within the ROM must start at the beginning address and extend from there. The original RPL configuration assumed just such a structure, so that converting the configuration from a linked sequence of ROMPARTs to a sequence of backup and library objects was relatively straightforward. The system determines the end of the sequence when it finds either an end-of-sequence mark, an object that is not a backup or library object, or an invalid checksum. In either of the last two cases, the user is warned of Invalid Card Data, but no further remedial action is taken.

Since RAM plug-in cards can be converted to the equivalent of ROM cards by simply changing a switch setting on the card, we decided that the structure of an unmerged plug-in RAM card should be the same as a ROM card, that is, a sequence of library and backup objects with the sequence starting at the lowest address of the card.

Ports. The RPL directory structure is one of the most tightly integrated aspects of the system. Having been conceived of as semipermanent storage which could dominate the use of free memory in a memory-limited system, it is implemented as a self-contained unit containing no pointers that need to be changed as other parts of memory change. In addition, it is relegated to the high-address end of free memory.

This highly integrated structure with no provision for referring outside itself precludes the inclusion of unmerged RAM cards as virtual subdirectories of the home directory. We decided to extend the mass storage analogy further and have separate data storage space locations analogous to flexible disk drives. Instead of drives A, B, and C, we have ports 1, 2, and 0. Ports 1 and 2 refer to the data contained in cards plugged into the corresponding plug-in slots. Port 0 refers to an area in built-in RAM that acts like a permanently plugged-in card (see Fig. 6). Unlike personal computer mass storage, however, the current drive is never any of these. The port specification must be included in the information given to any operation involving the ports.

The normal STO, RCL, and PURGE commands, which normally store, recall, and delete variables in main memory, are extended to allow transfer of information to and from the ports. Tagging a name argument with :0:, :1:, or :2: indicates to these commands that a port operation is desired. For example, if ABC is the name of an object in port 0, then :0:ABC RCL will return the object to the stack.

Since the only kinds of objects allowed in a plug-in data area are backup and library objects, any other kind of object is first encapsulated as a backup object. Similarly, RCL of one of the backup objects will pry the object out of the capsule.

Directory Management Extensions. The fact that the current drive is always none of the ports has several conse-

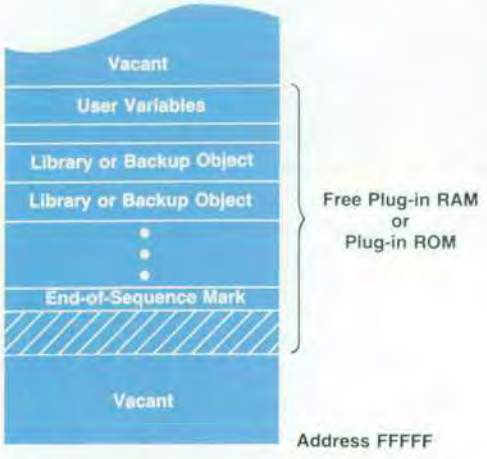


Fig. 7. *Layout of an unmerged (free) plug-in RAM card within the HP 48SX address space. An unmerged plug-in card is either port 1 or port 2 and has the same structure as port 0.*

quences. First, simply transferring a working set of related programs from a directory to a port will not result in a working set of programs in the port. This is because of the port-specific reference for names. If the program calls a subprogram by including its name, say SubProg1 EVAL, then only the directory is searched when the name is to be evaluated. On the other hand, if the subprogram is called by including a port-specific reference, say :0:SubProg1 EVAL, then only the specified port is searched. To compensate for this, we have included a "wild card" port specifier, :&:. The sequence :&:SubProg1 EVAL will search for the subprogram in all ports and then in the current directory before giving up. This calling sequence allows programs to be executed from directories or ports interchangeably.

A second consequence of there being no current drive is the lack of access to objects contained in directories stored in a port. Normally, the method of accessing an object in a directory is first to change context to the directory and then to use RCL or some other command. Since it is not possible to change context to a directory stored in a port, this method is not available without first copying the directory back to main RAM. This is solved by using a list as a complete path specifier for recalling a variable. For example, if A is a directory object stored in port 0 and it contains a variable B, then :0:{ A B } RCL will recall the contents of B to the stack. Similarly, it is possible to recall from any location without changing the context directory by using a list to specify the path completely.

Archive and Restore. With a method of mass storage available, it is natural to provide a means to archive the current contents of the calculator and later to restore this information. The operation ARCHIVE produces a copy of the entire home directory encapsulated as a backup object. It delivers this copy either to a specified port or to another machine.

Restoring from a backup copy of the home directory using the RESTORE operation reverses the archive process. Because of the potentially greater need for human intervention, RESTORE will not automatically restore from another machine. Instead, RESTORE will use a backup (or any other) copy of the home directory no matter how it was obtained.

RAM Recovery. With the large amounts of data that can be present in the machine, it is clear that additional data safeguards are necessary. One such safeguard is provided by the Recover RAM? operation.

Whenever it is found that the structural integrity of RAM has been violated, the user is given the opportunity to either start with a clean slate or attempt to salvage some data. If the user chooses to salvage data, the machine first searches through RAM, locating library or backup objects whose checksums are valid. It collects all of these into a new port 0.

It then searches for a directory object having the specific features of the home directory. If one is found, the RAM recovery operation verifies its structural integrity, and the operation is complete. To check the directory's structural integrity, the RAM recovery operation checks the structural integrity of each object within the directory (including recursively checking subdirectories) and removes any that are corrupt. If no home directory is found, the RAM recovery operation begins searching for ordinary directory objects. When it finds a directory it checks the structural integrity of each object within the directory (including recursively checking subdirectories) and removes any that are corrupt. The resulting corrected directories are named D.0, D.1, and so on, and are gathered together to form a new home directory, completing the recovery process.

Acknowledgments

The design, development, and testing of the HP 48SX firmware ultimately involved almost everyone in the Corvallis Division R&D department. In addition to the authors, those who contributed directly to the design and implementation were Ted Beers (application and interface management), Stan Blascow (BIOS and time management), Diana Byrne (plotting and EquationWriter), Gabe Eisenstein (EquationWriter), Grant Garner (decompile and ROM switching), Bill Johnson (MatrixWriter), Max Jones (menu system and editing), Paul McClellan (unit management), Pat Megowan (application interfaces), Nathan Meyers (I/O), and Bob Worsley (I/O and printing).

References

1. W.C. Wickes, "An Evolutionary RPN Calculator for Technical Professionals," *Hewlett-Packard Journal*, Vol. 38, no. 8, August 1987, pp. 11-16.
2. B.E. Musch, J.J. Wong, and D.R. Conklin, "Powerful Personal Calculator System Sets New Standards," *Hewlett-Packard Journal*, Vol. 31, no. 3, March 1980, pp. 3-12.
3. C.M. Patton, "Symbolic Computation for Handheld Calculators," *Hewlett-Packard Journal*, Vol. 38, no. 8, August 1987, pp. 21-25.

HP 48SX Interfaces and Applications

The HP 48SX scientific expandable calculator provides support for multiple applications, both built-in and externally developed, with customized user interfaces. The EquationWriter and interactive plotting are two of the built-in applications.

by Ted W. Beers, Diana K. Byrne, Gabe L. Eisenstein, Robert W. Jones, and Patrick J. Megowan

LIKE ITS PREDECESSOR THE HP 28S, the HP 48SX scientific expandable calculator is an RPN calculator designed as an electronic scratchpad for mathematical calculations. However, the simple user interface used in the HP 28S would have become overloaded if translated directly to the more capable HP 48SX. Consequently, the HP 48SX contains direct support for developing specialized user interfaces that can replace or extend the basic calculator interface. The support is used in the built-in applications such as the EquationWriter and interactive plotting, and is available for ordinary user programming and for externally developed applications. In this article, we will review the support mechanisms and give several illustrations of their use.

Managing Multiple Applications

Early in the development of the HP 48SX, it became apparent that without a common approach to application interface implementation, the calculator would not present a consistent methodology to the user. For example, when an application ends, it is important that the menu displayed before the application was started be restored. If one application restored the previous menu while another always displayed the MATH menu, user confusion would result.

Although a consistent approach to application interface design is important, so is the freedom of the designer to incorporate unique features that justify the need for a special interface. One of the challenges in developing the application interface engine for the HP 48SX was balancing consistency of operation with flexible design components. For the basic, stack-oriented, RPN operation of the HP 48SX, and for stack-oriented applications such as statistics, the user interface is handled by the built-in RPL outer loop. All other applications use an RPL tool called the parameterized outer loop, which is designed to customize a user interface.

The designer of an application can be expected to know how its interface should operate, but not necessarily to know or fully understand how the application should handle the application from which it was started or how to respond consistently to fatal error conditions and other unexpected events. The parameterized outer loop relieves the designer of these burdens while providing a common, robust method for handling application startup, application shutdown, and asynchronous event handling. The parameterized outer loop accomplishes this by handling the following major aspects of calculator operation:

- Saving the previous application's user interface
- Updating the application's display between key presses
- Waiting for and dispatching key presses, alarm interrupts, and unhandled errors
- Exiting the display and key handling loop
- Restoring the previous application's user interface.

Except for saving and restoring the previous application's user interface, the application-specific components of each step are specified by the application when it starts the parameterized outer loop.

Before the user can interact with an application such as the Equation Catalog, the application must set its user interface. The user interface is what makes the interaction with the application unique. For example, in the Equation Catalog, the familiar stack display is replaced by a list of equations, and the ▼ and ▲ keys no longer move the character cursor but instead move a list pointer around the equation list. An application sets these and other aspects of its interface when started by passing a set of user interface parameters to the parameterized outer loop. These parameters define how the application manages the HP 48SX display and keyboard and how the application interacts with the rest of the calculator environment.

Parameterized Outer Loop Operation

The operation of the parameterized outer loop can be summarized as follows:

```
Save the system or current application's user interface
If error in
   { Set the new application's user interface
     While exit condition object evaluates to FALSE
        { Evaluate display object
          If error in
             Read and evaluate a key
          Then
             Evaluate error handler object
        }
   }
Then
   Restore the saved user interface and error
Restore the saved user interface
```

The application specifies the unique operation components, such as the exit condition object and the display

object, when it starts the parameterized outer loop. This is how the application customizes the interface. The parameterized outer loop is responsible for the key-display loop, alarm interrupts, and low-level error handling.

Display Handling. There is no default display in the parameterized outer loop. The application is responsible for setting up the initial display and for updating it. The application display object is the method by which the application manages the HP 48SX display. This object, which is usually an executable program, can take advantage of the two main methods of displaying information that the HP 48SX supports: *passive* display update and *active* display update.

Passive Display Update. Passive display update involves using the display object to update any area of the display that needs to be changed after a key is handled. In this display handling model, each key is responsible for implicitly passing information to the display object regarding what areas of the display it hasn't changed. The display object then updates all other display areas.

Since the main outer loop itself uses this display update scheme, applications that use many standard keys, such as MatrixWriter, take advantage of the display update information passed by the standard keys to simplify their own display and key handling logic.

A major benefit of passive display update rests on the fact that the application programmer can make no display-related assertions at all in key handling, and still the display handling will work properly, albeit more slowly than necessary. As the application develops, the programmer can add assertions to those keys that do not affect certain display areas, thus saving time during display update. If the programmer misses a few combinations of key-display interaction, the application stills operates properly.

Active Display Update. The second method supported by the parameterized outer loop for handling display update is the more conventional active display update. In this model, each key that affects the display updates the display itself. With active display handling, the application display object can be reduced to a simple NOP (no operation). The major drawbacks of active display update are that all aspects of display handling must be considered by every key definition, and the implicit display update information required by other calculator resources must be determined whenever these resources are used by the application.

For consistency and robustness, most HP 48SX applications manage the display in the same manner as the main outer loop, namely with passive display update.

Hard Key Assignments. Any of the HP 48SX keys, in any of their six planes (unshifted, left-shifted, right-shifted, alpha-unshifted, alpha-left-shifted, and alpha-right-shifted) can be assigned for the duration of a parameterized outer loop application. The key object parameter specifies the keys to assign and their new assignments. In addition, there are two flag parameters that control how keys not assigned by the application are handled. If a key is not assigned by an application, and the allow default keys flag is TRUE, then standard or default key processing occurs, according to the do standard keys flag.

For example, if user keys mode is on and the key has a user key assignment, then the user key is processed if do standard keys is FALSE, or the standard key is processed if do standard keys is TRUE. If allow default keys is FALSE, then all nonapplication keys beep and do nothing else.

Menu Key Assignments. An application can specify any initial menu key assignments, in any of three planes (unshifted, left-shifted, and right-shifted), to be initialized when the parameterized outer loop is started. An outer loop parameter specifies the definition object for the application's menu, and may indicate that the current menu is to be left intact. When the outer loop is exited, the previous menu is restored automatically.

Since hard key assignments have priority over menu key assignments, it is possible to define more exotic behavior for the menu keys. To date, no parameterized outer loop application does so, however, since the menu key handling is very flexible and customizable itself.

Preventing Suspended Environments. Many applications need to allow arbitrary commands and user objects to be evaluated, but may not want the current environment to be suspended by the HALT and PROMPT commands. A parameterized outer loop flag specifies whether any command that would suspend the environment instead generates a HALT Not Allowed error. Since both HALT and PROMPT actually restart the main outer loop, which leaves the application suspended indefinitely without protection for its global resources, all current applications disallow suspension.

Nesting Applications. One of the powerful features of the HP 48SX is its ability to stack or nest multiple application user interfaces, effectively allowing an application to run within another application. For example, while working within MatrixWriter, one can press ↑ STK to start the interactive stack application to copy a value from the stack into MatrixWriter. Conversely, within the interactive stack, one can select a matrix and press VIEW to start MatrixWriter. In both cases, when the second application is finished, the first resumes where it left off. The parameterized outer loop makes sure all the details are sorted out.

Application Examples

The interactive stack is an HP 48SX application with which one can browse through the HP 48SX data stack and perform a set of stack-related operations based on the selected stack levels. Since the interactive stack is designed for stack operations only (including some object editing operations), it maintains strict control over the keyboard and display. This is accomplished with its key handling and menu objects. Unlike most applications, the interactive stack presents a different menu depending on how it is started. When an edit line does not exist, a full menu of operations is displayed. When an edit line does exist, the interactive stack displays a more restrictive menu, reflecting the fewer operations available. To implement this difference, the interactive stack passes one of two menu objects to the parameterized outer loop as its menu specification.

MatrixWriter is an HP 48SX application that simplifies the entry of matrix objects. Like the interactive stack, MatrixWriter controls certain keys that are redefined for its environment, such as ▲. Unlike the interactive stack, however, MatrixWriter allows all undefined keys to operate normally, since many standard key definitions, such as +, are useful in MatrixWriter.

Both the interactive stack and MatrixWriter use the passive display update method for managing their output. In the case of MatrixWriter, this is especially useful and important, since the standard edit line interface is used extensively within the MatrixWriter environment.

Customization by the User

The standard keyboard and display of the HP 48SX are designed for general use, offering direct access to numerical computation and indirect access to other features. For users who want direct access to features of their own choosing (or creation), the HP 48SX has a number of tools for customizing the user interface. The user can redefine keys, define a custom menu, customize how key definitions are executed, and maintain a variety of customized environments.

In the HP 28S, key definitions are objects of a special form. For easier customization, we changed the HP 48SX so that any object can be a key definition. For example, the user can assign the string 5 and the function + to keys, and those keys will act the same as the normal **5** and + keys.

For each key, the user can assign an object in one of six key planes: keys can be unshifted, left-shifted, or right-shifted with alpha on or off. If key assignments are viewed as yet another shift, this makes 12 key planes in all. The user can enable or disable the current assignments by pressing **↰USR**, or by setting or clearing a flag. The assignments can be recalled as a list of alternating key codes and objects, and such a list can be used to make assignments.

Another way to define keys is the custom menu. After storing a list of objects in a variable named CST, the user can press **CST** to put the first six objects in the menu, **NXT** to put the next six objects in the menu, and so on. This method doesn't involve the key assignments described above; rather, it uses the standard key definitions that make the menu system work.

The objects in the custom menu are given the same shifted interpretations as in built-in menus. For example, a name is executed, stored into, or recalled, depending on whether the key is unshifted, left-shifted, or right-shifted, just as in the VAR menu. Units are multiplied, converted, or divided, just as in the UNITS menu. Alternatively, the user can specify separate objects for the menu label and for unshifted, left-shifted, and right-shifted actions.

The most radical customization is called vectored ENTER. When the user presses a key in normal operation, the corresponding object is either written to the command line or executed. In the latter case, the text already in the command line must be parsed and executed first, and then the key-definition object is executed. The user can customize two steps in this process by storing programs in variables αENTER and βENTER.

The program in αENTER takes over parse-and-execute responsibilities. Such a program might either (1) print the command-line text and then execute OBJ→, which parses and executes as usual, (2) modify the text and then execute OBJ→, or (3) parse the text itself.

After the key-definition object is executed, its text form is given to βENTER as an argument. Such a program might print the text and the contents of the stack, drop the text and modify the results on the stack, or display status information or otherwise prepare for the next input from the user.

Vectored ENTER is enabled by setting both its own flag and the flag for user key assignments. The latter condition allows the user to disable vectored ENTER from the keyboard. This safety feature is important, since faulty customization routines can totally disrupt calculator operation.

Finally, the user can maintain a variety of interfaces customized for different purposes. Since the custom menu and vectored ENTER are defined by variables, switching directories can cause the interface to change accordingly. On the other hand, key assignments are independent of the directory. By assigning directory-switching programs to keys, the user can readily switch from one interface to another.

The EquationWriter

The primary design objective of the EquationWriter was to overcome several factors limiting the ease of use of existing calculators. The EquationWriter is the first application to emerge from advances in display technology, both hardware and software, compared with the HP 28S. The general result of these advances can be seen in the inclusion of the graphic object data type and the virtual screen of the HP 48SX.

The basic idea of the EquationWriter is to show mathematical expressions as they appear in textbooks or as normally written by hand—for example:

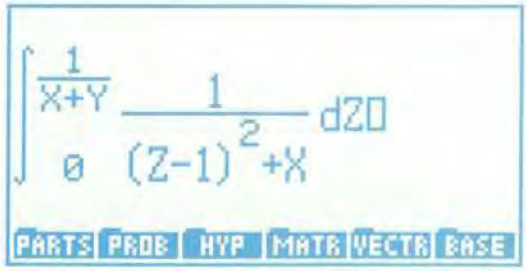
- Numerators above denominators, separated by a horizontal line
- Exponents written as superscripts, in a smaller font
- Parentheses of adjustable height
- The use of standard symbols for integral, summation, etc.

This in itself is novel only in the calculator world. However, the main challenge, which was felt not to have been met even by existing desktop systems, was to come up with a consistent and intuitive way of producing and modifying these formatted displays as the user enters the expression, symbol by symbol.

The most obvious limitation on the entry and display of mathematical expressions in the standard linear format is that when they get even moderately large, it becomes extremely difficult to survey the subexpression groupings visually and sort out all the parentheses. An expression like

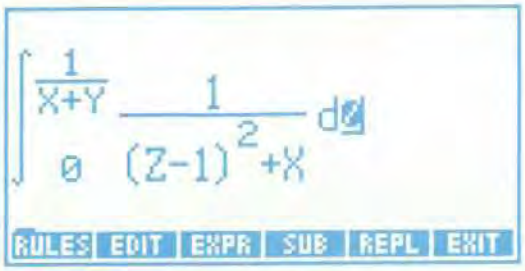
$$\int(0,1/(X+Y),1/(1+((Z-1)\hat{}2+X)),Z)$$

is terribly tedious to read and understand, compared to

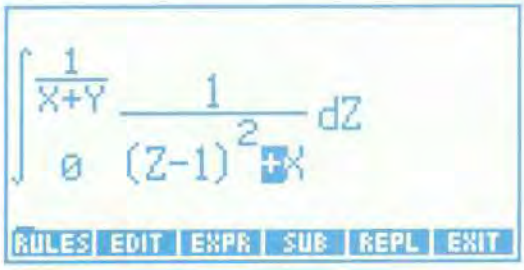


This problem was especially onerous for the HP 28S FORM interface (renamed RULES in the HP 48SX), which applies operations like commutation, association, and distribution to subexpressions of a given expression. Locating

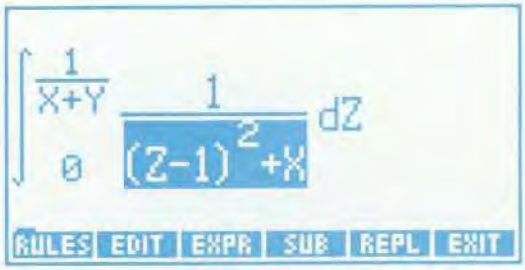
the desired subexpression (which very likely did not even fit on the screen) amid the plethora of parentheses, and recognizing its relation to the likewise messy and stretched-out result, was too much for many users to bother with. In the HP 48SX, the RULES interface is a subsystem of the EquationWriter, which can be entered any time the expression typed so far is complete (a+b is complete, a+ is not) by pressing the ◀ key. This sends the cursor back to the rightmost object (number, variable, or operator) in the expression, appearing as an inverse-video highlight of that object.



From here the highlight-cursor can be moved around with the arrow keys. Pressing ◀, ▼ results in:



The subexpression selected for an operation is that which is included in the range of a highlighted operator (if a variable or number is highlighted, the subexpression simply consists of that object alone). A menu key toggles between highlighting the individual object and the selected subexpression.



This removes any remaining uncertainty about which subexpression is selected (although it is not usually needed because the grouping is so much more apparent, and more of the expression tends to fit on the screen).

Since the subexpression selection mechanism was included for the RULES interface, it made sense to allow editing of subexpressions as well. Once a subexpression is selected for editing, a command line is brought up in which to modify the subexpression in its normal string form. This is mainly useful for changing the spelling of a name or number. Other means of modifying and combining expressions are provided by the SUB (substitute), REPL (replace) and RCL (recall) functions: the first sends a copy of the selected subexpression out to the data stack, the second replaces the selected subexpression with the algebraic object on the data stack, and the third inserts the object from the stack in the cursor position when in entry mode (rectangular cursor showing). Of course, it is possible to back up with the normal backspace key (⌫).

Another problem with the standard linear format is remembering the meaning and order of multiple parameters. A frequent complaint about the HP 28S was that no one could remember how to type in the parameters to the INTEGRAL function. Did the lower or upper bound come first? Did one have to type the d that goes with the variable of integration? In the EquationWriter there can be no such confusion. Upon pressing the ∫ key, one immediately sees an integral sign, with the cursor in the position of the lower bound.

Any expression can be entered as the lower bound; it is terminated by the ▶ key, which is the general means of terminating any syntactic piece (exponent, numerator, denominator, etc.). The cursor then moves to the upper bound position.

If one of the subexpressions grows vertically (e.g., when entering a quotient for the upper bound), the integral sign stretches to accommodate it.

After the upper bound and integrand are terminated in turn, a d appears with the cursor to its right, making it obvious that a variable name is now required.

This is a good place to mention another significant feature of the EquationWriter, which is its real-time syntax checking. With the cursor in the variable of integration position, the system will not accept any input except a legal variable name. Pressing + here will immediately result in the Invalid Syntax message, with the cursor returned to the left of the d. Similar behavior results from following a prefix function immediately with an infix function, and so forth. The EquationWriter uses the same internal parsing engine that is used to parse algebraic expressions typed into the command line. All graphical events in the EquationWriter (putting up a new symbol, inserting punctuation, altering the sizes of parts of the picture, and repositioning the cursor) are triggered by transitions across syntactic boundaries, as interpreted by the internal parser. The ▶ key always has the meaning of "go to the next syntactic position", so that it not only terminates exponents, denominators, and so on, but also results in the insertion of any required token following the current position, such as a closing parenthesis, or a comma if you are in the first argument of a function requiring two or more arguments. If the current syntactic piece is not legally completed, the cursor will not advance.

This general use of the ▶ key was actually quite controversial during the early phases of development, and this illustrates the challenge of coming up with an intuitive entry procedure, as mentioned above. One objection was that the ▶ key is an unnecessary nuisance when entering a typical polynomial: after typing the 2 in an expression like ax^2+bx+c, why can't I just type + ? There was no problem in adopting such a rule, based on operator precedence, but the result would be that the hated parentheses would start sprouting any time the exponent, numerator, denominator, or other expression was not typical (i.e., simple), and the user would have to remember to type the parenthesis before starting the subexpression (just like in the old linear format). In the end it was decided to provide both methods as modes that can be toggled by the user. A similar problem with respect to division was solved by providing, in effect, two division operators: one prefix (press ▲ to initiate a complex numerator) and one infix (press ÷ to draw a line under the preceding subexpression, going back to an operator of lower precedence than division). However, the uniformity of the ▶ key has proven to be a contribution to an intuitive interface. Not only do all built-in operators work similarly, but also all future operators, with their own distinctive graphical properties defined by users who write libraries, will also have the same feel.

The ideal of displaying expressions just as they appear in textbooks turned out to be unattainable, mainly because textbooks were found to follow different rules and ill-defined conventions. In the expression ax^2+bx+c, everyone assumes that a and b are coefficients, but in general, variable names cannot be limited to one character, nor should there be something special about the letter x. Thus we gave up the idea of incorporating implied multiplication in the display. However, it is present in the entry rules: typing 2A automatically creates the display 2*A, and similarly, any sequence of two contiguous tokens functioning as operands results in the insertion of the multiplication symbol. The display is unambiguous, but the typing is simplified.

Graphics and Plotting

Scientists and engineers use graphics for many aspects of their work: describing problems, studying functions, working out solutions, presenting data, and so on. Our main goal for the HP 48SX graphics and plotting system was to offer plotting tools beyond those of the HP 28S, which provided function plots and statistical scatter plots. We also wanted to contribute to the overall goal of making the calculator easier to use. A third goal was better integration of graphics with other capabilities of the machine.

Our design choices were based on feedback from HP 28S users, guidelines provided by the National Council of Teachers of Mathematics, consultations with mathematics educators, and the experience of team members as college instructors, mathematicians, physicists, and engineers. The result is the HP 48SX graphics and plotting system, which has the following new elements:

- A new RPL object type called a graphics object, or GROB, along with commands to create and modify GROBs
- An enhanced interactive environment for plotting and graphics
- Four new mathematical plot types and two new statistical plot types.

The HP 48SX uses a new object type called a graphics object, or GROB, to represent graphical images. Like all objects, GROBs can be placed on the stack, included in programs, stored in variables, and exchanged with other calculators. Most graphics commands act on the GROB stored in a special display region called PICT. The HP 28S used one area of memory for all displays. The addition of a separate graphics display area in the HP 48SX simplifies mixing graphics and stack operations. Both display areas are expandable, with scrolling available to view GROBs larger than the display.

Commands are available to draw geometric shapes, sketch freehand, or do cut-and-paste operations with smaller GROBs. Most of these commands are available in both interactive and programmable forms. For example, geometric shapes include boxes, circles, and lines:

Freehand sketches include arbitrary curves and individual pixels:

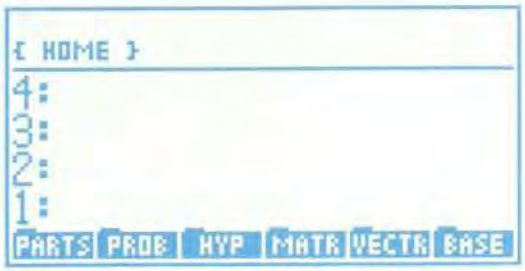
The REPL (replace) command takes a GROB from the stack and pastes it into the PICT GROB. The next example includes a label made from a string (by the →GROB command) and a picture imported from a computer.

Adding a modify step to cut-and-paste leads to a rudimentary but entertaining form of animation.

The calculator itself uses GROBs for all display-related tasks. Graphic applications such as plotting use GROBs, of course, but text must also be converted to pixels before it can be displayed. For example, the components of the normal display (status information, stack objects, command line, menu labels) are created as individual GROBs and then pasted onto the GROB in the stack display area. Applications such as EquationWriter and MatrixWriter similarly construct and combine GROBs.

Much of the benefit of GROBs, like other object types, is in having standard tools for standard objects used by both the system and the user. This uniformity leads to smaller code with fewer defects.

An example of the internal structure of a GROB can be seen in the PARTS menu label in the MATH menu. The calculator creates a small GROB for this label and then pastes that GROB onto the larger GROB for the whole display.



The command-line form of this GROB is:

```
GROB 21 8 E30000FFFFF11B9131555BD1119BB1D55B71D55B91FFFFF1
```

where 21 and 8 are the width and height in pixels, and the hexadecimal digits represent the graphical data, starting left to right across the top row:

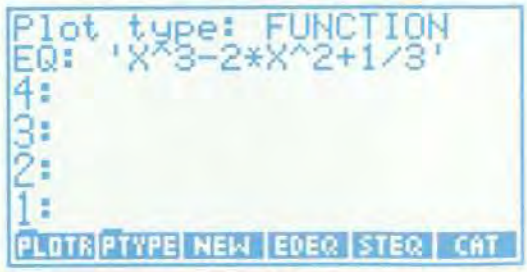
```
E30000
FFFFF1
1B9131
555BD1
119BB1
D55B71
D55B91
FFFFF1
```

Each hexadecimal digit represents a horizontal sequence of four pixels, with the least-significant bit representing the leftmost pixel. For example, the hexadecimal digit E, written as 1110 in base two, represents four pixels: off, on, on, on.

Each row is represented by an even number of hexadecimal digits because the display hardware reads one byte (two hexadecimal digits) at a time. This requires up to seven bits of padding at the end of each row; in this example, three bits of padding are required.

Function Plots

Like the HP 28S, the HP 48SX uses the variables EQ (equation) and PPAR (plot parameters) to control plotting. The user can maintain multiple plotting environments by creating multiple directories, each with its own EQ and PPAR. When the user presses ↰ PLOT, the plot application first shows the current equation (EQ) and plot type (one of the plot parameters):



We first demonstrate the plot type FUNCTION, which is an enhanced form of the HP 28S plot type. Later we will show the results from other plot types.

The user can press NEW to name a new equation, EDEQ to edit the current equation, or CAT to show a catalog of equations:

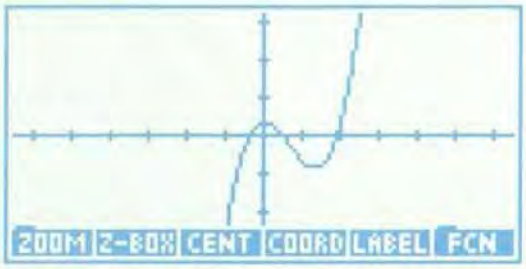
The equation specified by EQ can be an expression, an equation, a program that computes values, or the name of a variable that contains one of these. Multiple equations can be plotted simultaneously by combining them into a list and storing that list in EQ.

After selecting the equation, the user can press PTYPE to change the plot type. Other plot parameters control the plot's scale and the placement and labeling of the axes. To change the other plot parameters, the user presses PLOTR:

Like the initial plot menu, the PLOTR menu displays the current values of relevant variables. To avoid interference with normal stack activity, these displays are maintained only as long as the commands in the menu are being used interactively.

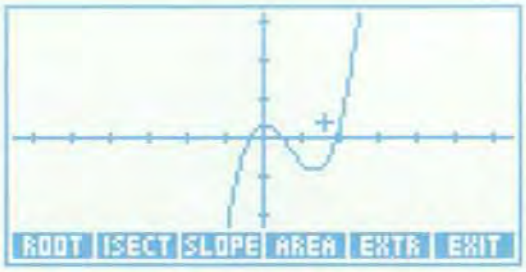
When the plot parameters are set, the user can press ERASE to clear PICT, or skip this step to superimpose the plot on the current PICT. The plotting is started by pressing DRAW or AUTO; the latter attempts to scale one or both axes automatically, according to the plot type.

When the plot is completed, a menu of interactive operations appears:

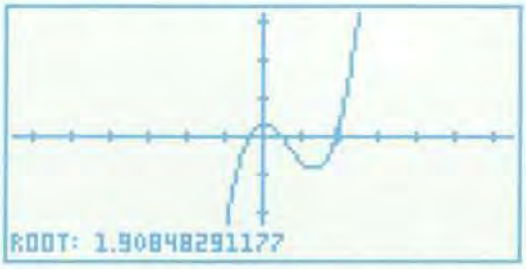


In the center of the display is a cross-shaped cursor, which the user moves by pressing the arrow keys. The cursor is used to specify locations for a variety of plotting and graphics operations. Pressing COORD causes a display of the cursor coordinates to replace the menu labels. If the PICT GROB is larger than the display, the user can force the display to scroll by moving the cursor off the edge of the display.

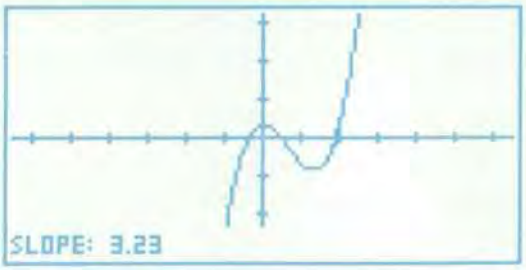
If the plot needs adjusting, the user can zoom in or out along either or both axes, or define a new center. If the plot is satisfactory, the user can press FCN to show a menu of mathematical tools (applicable only to the FUNCTION plot type):



With these tools the user can analyze the function without leaving the interactive graphics environment. For example, pressing ROOT invokes the solver to find the nearest root (the cursor moves to the root):

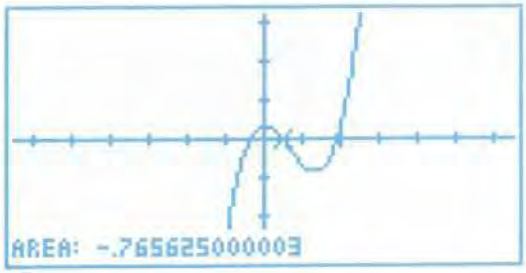


Pressing SLOPE invokes differentiation to find the derivative at the cursor's location:



Pressing EXTR invokes differentiation and the solver to find the nearest extremum (the cursor moves to the extremum):

Pressing AREA (twice, with the cursor at each limit) invokes numerical integration:



When EQ contains of list of equations, the user can apply these tools to any individual equation, or use ISECT to find the intersection of any pair of neighbors in EQ.

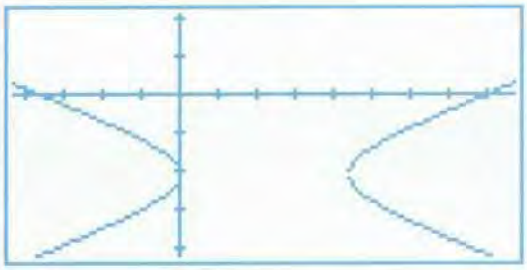
Other Plot Types

To the HP 28S plot types FUNCTION and SCATTER the HP 48SX adds mathematical plot types CONIC, POLAR, PARAMETRIC, and TRUTH, as well as the statistical plot types HISTOGRAM and BAR. The mathematical plot types share code for the basic steps: setting up the plotting environment, assigning successive values to the independent variable, evaluating EQ, plotting the corresponding points, cleaning up the environment, starting the interactive phase, and handling errors. Each mathematical plot type requires its own code to process EQ once at the start and to process each result of evaluation.

CONIC plots handle circles, ellipses, parabolas, and hyperbolas. This type turned out to be a simple combination of existing tools: the code underlying the command QUAD is used to turn EQ into two branches. Then the code in the FUNCTION plot type that plots both sides of an equation is used to plot both branches of EQ. For example, the equation

$$4\cdot X\hat{}2-9\cdot Y\hat{}2-24\cdot X-90\cdot Y-225$$

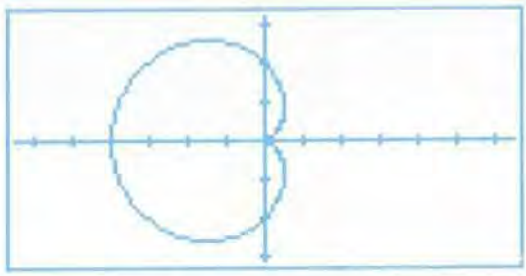
is plotted as:



POLAR plots show the independent variable as a polar angle and the dependent variable as the radius. For example, the equation

$$2\cdot(1-\mathrm{COS}(X))$$

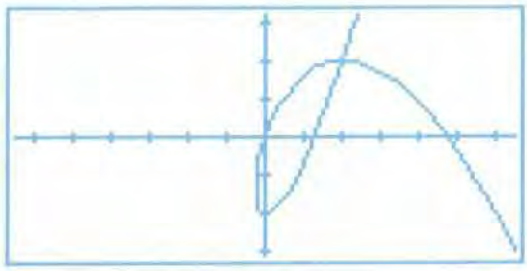
is plotted as:



PARAMETRIC plots show a complex-valued function of one real variable, where the real and imaginary parts are functions of the independent variable. For each point, the horizontal coordinate is given by the real part and the vertical coordinate by the imaginary part. For example, the equation

$$T\hat{}2-T+i\cdot(T\hat{}3-3\cdot T)$$

is plotted as:



TRUTH plots show truth-valued functions of two real variables. The location of each pixel represents the domain, and the value of the pixel represents the function value. Often the truth-valued function is the compostion of a function of interest, such as a real function of two variables or a complex function, and a projection function that maps function values to truth values. For example, consider a two-argument function. Plotting the expression

$$(2\cdot X\hat{}2-3\cdot Y\hat{}2+X\cdot Y)\ \mathrm{MOD}\ 16>8$$

produces a contour plot of the polynomial with contour intervals of 8:

A second example is a complex function. Plotting the expression

$$\mathrm{SIGN}\ (\mathrm{RE}(Z\hat{}3-2\cdot Z))==\mathrm{SIGN}\ (\mathrm{IM}(Z\hat{}3-2\cdot Z))$$

where Z is defined to be $X+i\cdot Y$ produces a quadrant plot of

the polynomial. The black regions are the points mapped to the first or third quadrants of the complex plane and the white regions are points mapped to the second or fourth quadrants.

HP Solve Equation Library Application Card

The card contains a library of 315 equations, the periodic table of the elements, a constants library, a multiple equation solver, a finance application, and engineering utilities.

by Eric L. Vogel

HISTORICALLY, EVERY HP programmable calculator has had programs available in some form. There are application books with printed programs and keystroke sequences for machines like the HP 55, 25, 33E, 11C, and 32S, application pacs with programs on magnetic cards for the HP 65 and 67, and pacs with programs in plug-in modules for the HP 41 and 71. These books and pacs focus on computation-intensive (as opposed to data-intensive) problems in specific science or engineering disciplines. Program size and capability are limited primarily by available memory and the single-line calculator display.

The HP Solve Equation Library application card provides this capability for the HP 48SX scientific expandable calculator, but without the limitations of previous pacs. The focus on computation-intensive solutions is preserved, but for a wider range of disciplines than in individual pacs in the past. An additional focus on data-intensive applications has been added in the form of on-line, electronic reference information (two thirds of the card contains data). The 128K-byte memory capacity of the card makes these two emphases possible, and the large display allows improved user interfaces for the interactive applications.

The card contains six major applications:

- Equation Library (Fig. 1). This is the primary application for which the card was named: a collection of 315 equations organized in a catalog of 15 different subjects, each containing a catalog of equation titles. For each title, the user can examine the equations and catalogs of names, descriptions, and SI or English units for its variables. A key contribution is pictures that describe the physical situations represented by the equations. Our goal was that the subject, title, and reference information would help a user select an equation to use with the HP Solve

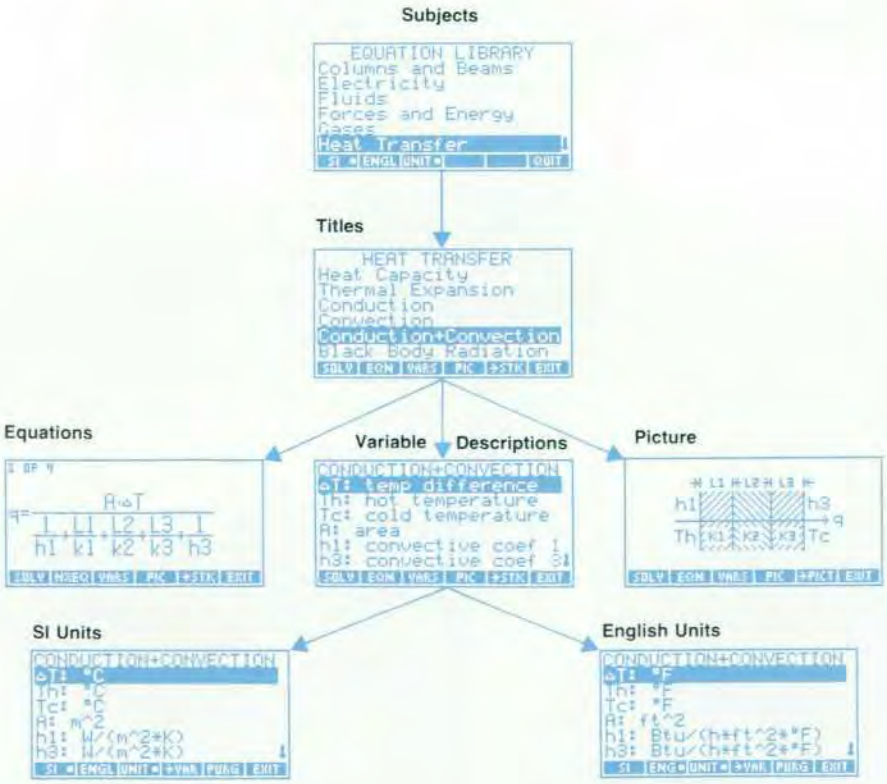


Fig. 1. *Equation Library user interface.*

application or the card's Multiple Equation Solver (discussed below).

- Periodic Table (Fig. 2). This application contains all the chemical data (such as atomic weight and density) that appears on a standard periodic table of the elements. The primary user interface is the universally recognized grid of elements. The user can move a highlight block to see any element and its most-used properties on the grid. There is also a catalog of 23 properties available for each of the 106 elements. Properties can be plotted versus atomic number to reinforce the relationship between property and atomic structure. A molecular weight calculator allows typing chemical formulas and quickly calculating their molecular weights.
- Constants Library. This is a collection of 39 commonly-used physical constants. These appear in catalogs of symbols, descriptive names, values, and SI or English units.
- Multiple Equation Solver. This is a collection of commands that make it possible to use the Multiple Equation Solver to interact with the user's own equations as a group, rather than just the groups of equations from the Equation Library.
- Finance. This application duplicates the basic calculations performed by HP financial calculators: time value of money (the relationship between the number of payments, interest rate, present value, payment, and future value) and amortization.
- Engineering Utilities. These are engineering functions that support the computational needs of some of the equations in the Equation Library.

These applications come in two forms: interactive for working with the application and its data, and noninteractive for programmatic calculations and access to the on-line data.

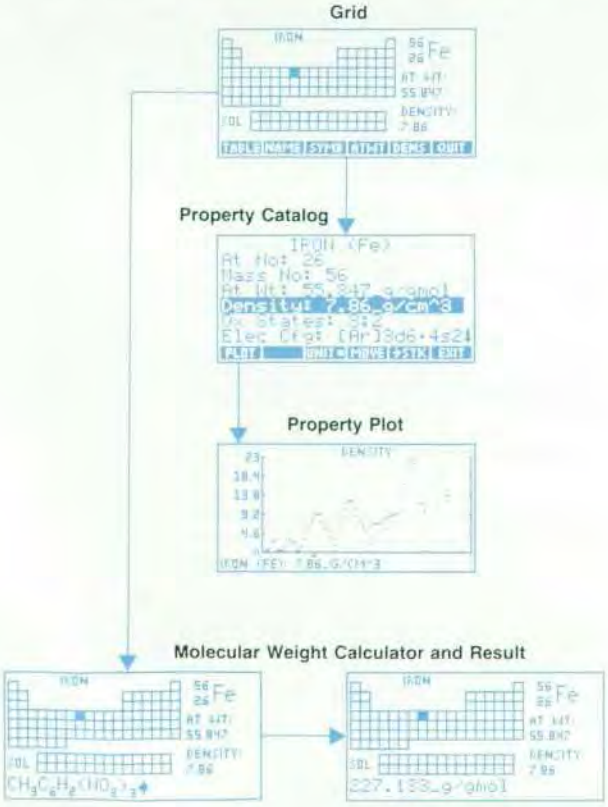


Fig. 2. *Periodic Table user interface.*

Equation Library Evolution

The Equation Library concept stemmed from three observations. First, students need a wide variety of solutions because of the number of classes they take. Because application pacs are limited to specific areas, students often need several pacs to cover the different disciplines they study simultaneously. Second, most of the engineering applications for the HP 41 and its predecessors are programs that simply solve an equation for a specific variable. More sophisticated programs of this type allow interchangeable solutions in which most or all of the unknown variables can be calculated as long as they can be isolated algebraically in the equation. Some programs use iterative techniques to find a solution when an algebraic isolation is not possible. Later application pacs attempt to allow the user to select different units for the different variables.

Third, the HP Solve application in the HP 48SX takes an equation and makes it into a small, self-contained application. It solves for any variable given the others, allows units to be specified for each variable, handles unit conversions automatically, and provides a consistent, straightforward user interface for interacting with all the variables.

From these three observations, we realized that we could create a collection of small applications in most of the science and engineering disciplines of previous application pacs by combining a collection of equations with HP Solve. The HP Solve user interface allows each application to work the same way, and the ability to solve for any variable and automate unit conversions makes these applications more versatile than in previous application pacs.

Interacting with Groups of Equations

As the equation selection proceeded, we found that related equations were usually needed as a group, rather than independently (Fig. 3). While there are certainly instances where only one of the equations is needed, more often the entire set is used to find the value for a particular variable. To provide simplified access to related equations, we group them together under a single title, such as Linear Motion or Ohm's Law and Power, rather than forcing the user to return to the Equation Library to select each equation individually.

When we examined how a user typically interacts with a group of equations, we realized that the solution proce-

$$x = x_0 + v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 \qquad V = I \cdot R$$

$$x = x_0 + v \cdot t - \frac{1}{2} \cdot a \cdot t^2 \qquad P = V \cdot I$$

$$x = x_0 + \frac{1}{2} \cdot (v_0 + v) \cdot t \qquad P = I^2 \cdot R$$

$$v = v_0 + a \cdot t \qquad P = \frac{V^2}{R}$$

(a) Linear Motion (b) Ohm's Law and Power

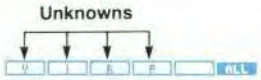
Fig. 3. *Examples of common equation sets.*

dure is straightforward, but gets tedious as the number of equations and variables gets large. Here is the manual process for finding all the variables for a set of equations given that some of them are known:

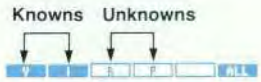
- Use the known variables to select an equation containing only one unknown.
- Solve for the unknown.
- Add the variable just calculated to the set of known variables.
- Use the combined set of known and calculated variables to select another equation containing only one unknown.
- Solve for the unknown.
- Repeat this process until either all the unknowns have been found or as many unknowns as possible have been found from the given set of knowns.

To make using the equations more straightforward, we have automated this manual select-and-solve process by developing an extension to HP Solve called the Multiple Equation Solver (MES). The MES selects the appropriate equation to solve based on whether it has one remaining unknown, and then solves for that unknown using the same numerical root finder used by the HP Solve application. It tracks the variables that have been solved for, and uses different equations to calculate other unknowns as soon as there is enough information available.

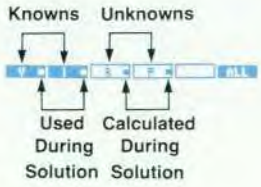
The barrier to proper functioning of the MES was identifying whether a variable is known or unknown. The existence of the variable alone is not sufficient—after a solution has been determined, all the variables exist. The key underlying principle is that the state of a variable (known or unknown) is independent of the value of the variable. The MES uses this state information to select the equations to be solved and the order in which to solve them.



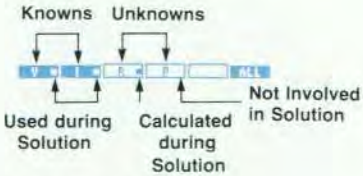
(a) Initial menu for Ohm's Law and Power



(b) After entering values for V and I



(c) After solving for all unknowns



(d) After solving for R only

Fig. 4. *Multiple Equation Solver menu key appearance.*

Displaying Variable States

The MES user interface is similar to that of HP Solve. A menu of variable names is displayed in the menu key area at the bottom of the display. The appearance of the menu keys is used to distinguish the MES state information. An extra key, ALL, appears at the end of the menu.

Initially all the menu keys are white with black letters (like HP Solve), indicating that all of the variables are unknown (Fig. 4a). Typing a value and pressing a menu key stores the value in that variable and changes the key to black with white letters, indicating that the variable is known (Fig. 4b).

Pressing the ALL key solves for all remaining variables, or as many as can be found from the given set of knowns.

Messages appear during the solution identifying which variable is being solved for and its resulting value. After the solution has completed, each variable retains its initial state. Correspondingly, each menu key retains its initial appearance. Black keys (knowns) remain black, and white keys (unknowns) remain white. This simplifies solving a problem using the same knowns and unknowns but with different values.

Indicating Variable Relationships

After a solution, some of the menu keys will have a small block in them to indicate the roles their variables played in the solution (Fig. 4c). A block in a black key (known) indicates that the variable was used to find an unknown in a particular equation. A block in a white key (unknown) indicates a value was calculated for that variable during the solution. This represents a unique state for a variable—it is an unknown, yet it has a calculated value. The next time this variable is solved for, this calculated value will be used as its initial guess.

Pressing the shift key followed by a menu key solves for that specific variable, regardless of whether it is black or white (known or unknown). After a variable is solved for, its menu key is shown in white with a block to indicate an unknown that was solved for with a calculated value (Fig. 4d). Other menu keys may have blocks in them based on the roles their variables played in the solution.

Solution Summary

A summary of the solution procedure is available by pressing the shift key followed by the ALL key (Fig. 5). This summary shows a catalog of each unknown that was found,

(a) Values Calculated during Solution

(b) Equations Used during Solution

Fig. 5. *Solution summary.*