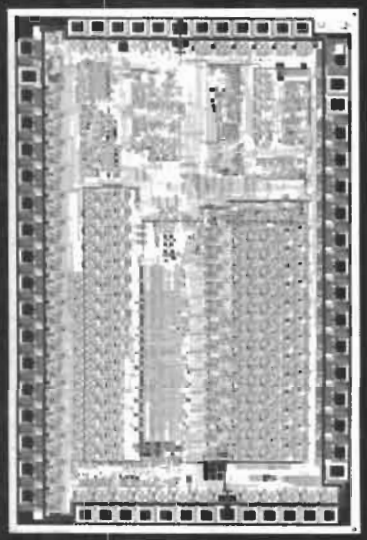
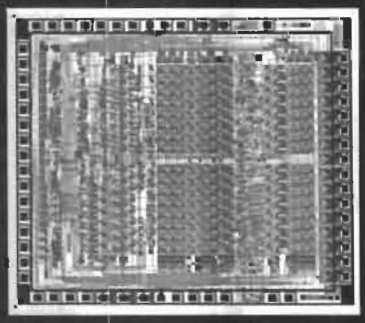
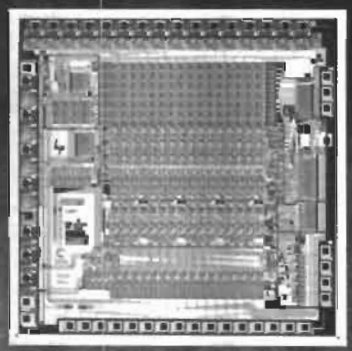
JUNE 1979

HEWLETT-PACKARD JOURNAL







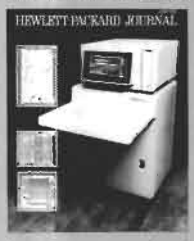
HEWLETT-PACKARD JOURNAL

Technical Information from the Laboratories of Hewlett-Packard Company

JUNE 1979 Volume 30 • Number 6

Contents:

In this Issue:



This month's issue is devoted entirely to the HP 300 Computer. This compact, moderately priced computer has generated more excitement within Hewlett-Packard than we've seen for a long time. Designed for business data processing, it represents the state of the art in both hardware and software and sets formidable new performance and convenience standards for computers in its class.

A single HP 300 will support up to sixteen terminals handling inventory control, accounts receivable processing, and other business applications. Of course, like all computers, the HP 300 has to be programmed to do these things, and since HP doesn't supply the necessary business applications programs at this time, the first customers for the HP 300 are expected to be software companies, original equipment manufacturers (OEMs), and sophisticated end users, who will use the advanced software HP does supply to develop business programs for themselves and their customers.

There are so many technical contributions in the HP 300 that there isn't room to cover them all in one issue. For that reason, we're devoting two consecutive issues to a single product for the first time ever. This month we describe the HP 300 as the user sees it, starting with an introductory overview (p. 3). This is followed by articles on the keyboard and display (p. 6), what it's like to develop programs on the HP 300 (p. 9), facilities for storing and retrieving data (p. 16), the two high-level programming languages now available, RPG (p. 20) and BASIC (p. 23), and the package design (p. 26). Next month we'll go inside and look at the design of the hardware and the operating system.

This month's cover photo shows the HP 300 system unit with the three specially designed silicon-on-sapphire integrated circuits that are used in its processor section. Silicon-on-sapphire technology is a major contributor to the HP 300's remarkable combination of compactness and computing power.

-R. P. Dolan

A Business Computer for the 1980s

A totally new business-oriented design based on HP's silicon-on-sapphire integrated circuit technology, this new system packs a vast amount of processing power into a surprisingly small package.

by George R. Clark

HEWLETT-PACKARD'S new HP 300 Computer, Fig. 1, is an advanced, office-oriented multi-user system designed to simplify the tasks of developing and running business application programs. The product of one of the largest development programs ever undertaken by Hewlett-Packard, the HP 300 marks the beginning of a major new computer family intended for business data processing applications. This issue and next month's issue of the Hewlett-Packard Journal will review the HP 300, beginning in this article with a discussion of the design considerations and an overview of certain features that distinguish the HP 300 from other computers.

Design Requirements

Today's computer must above all be easy to use. This is obviously a desirable characteristic for computers in all applications, but it is especially important in the business environment. If the computer is to be effective in managing the day-to-day problems of order entry, inventory control, and accounts receivable, people having limited technical skills, such as clerks and secretaries, must feel comfortable using it. These persons must be able to run the application programs and routinely access data bases in a natural manner without having to remember complex procedures or long command sequences. Simply, the computer must be viewed as a friend, not an adversary.

The system should provide quick response to inquiries from its users. It should be possible for a salesperson to respond to a customer's inquiry and ascertain the scheduled shipment date of an order while the customer waits on the phone. High system performance and a powerful software architecture are required to support such transactions effectively while several other users are updating data bases, accessing files, writing reports, or developing programs at the same time. To be well-suited for the office environment, the system should consume minimal floor space and require no major changes to office facilities for installation. This means that no special temperature and humidity controls and no special power should be required. The system must be quiet, safe, and esthetically pleasing. There must be no radiated electromagnetic energy that would interfere with other equipment in the vicinity, and static discharges to the machine produced by walking on carpets during periods of low humidity must not cause malfunctions.

As the needs of the customer grow and the use of the computer increases, the system must be expandable to provide enhanced performance and new application capabilities. If a sales manager buys a system to handle incoming orders and later realizes the need to publish reports and

Fig. 1. *An HP 300 Computer System including the system unit (center), a printer, two application terminals, and an additional disc drive. Up to 16 terminals, two printers, and 260 megabytes of disc storage can be supported by one system unit.*

coordinate the orders with inventory control and the shipping/receiving department, it shouldn't be necessary to throw away the existing investment in hardware and data base development. Rather, it must be possible to add the necessary peripherals, applications programs, and memory with minimal cost and system down-time.

With increasing dependence upon the system to manage more and more work, reliability becomes critical. The responsibilities assigned to a business computer can increase to such as extent that there really is no effective backup procedure when a system failure occurs. For example, the volume of billing notices issued by a lending institution is often so large that the job cannot be completed without the computer's assistance. Yet, a delay in mailing the bills can have devastating financial effects for the company. Thus, the potential benefits of using a computer in the business can easily be nullified by untimely failures and costly repairs.

The total cost of owning the system must be low. This requirement is really a result of proper achievement of the other goals, but deserves mention separately. Ease of use at the cost of undue complexity, reliability through the use of special, costly hardware, quick repair via a force of on-site maintenance personnel, expandability at the expense of difficult reconfiguration procedures, low electromagnetic radiation through exotic packaging techniques, performance at the expense of power consumption—these design philosophies, if followed, would drive up the total cost to the customer and prevent the computer from achieving widespread acceptance in the business market.

Such considerations formed the basis of the HP 300 design objectives, with the result that many departmental data processing problems now may be solved in a manner never before practical.

The HP 300 Approach

From the moment power is applied to the HP 300, its ease of use is striking. The computer comes up running and ready to use with no complicated cold-load procedures. The operator may imediately begin developing programs, activating existing applications, accessing files, and so on. Various questions the operator may have about commands, procedures, or syntax may be answered through the on-line reference "manual" called HELP. A simple English question like, "HOW DO I BUILD A FILE?" causes the system to respond with the appropriate information—in this case, instruction on use of the CREATE FILE command. The operator soon notices that spelling errors are corrected and commands may even be abbreviated. This allows the operator to concentrate on being productive, rather than mentally battling the computer.

The novel features of the integrated display system (IDS) exemplify the friendliness and sophistication of the HP 300 System. The display consists of several *windows* (independent display areas) that may be used by the operating system or an application program to display information from several different sources at the same time. For example, an inventory control program might use one window to display the parts list while another window gives access to vendor information. Along the right side of the IDS are eight pushbutton switches whose functions may be dynamically altered to suit the needs of the system and user. These *softkeys* allow high-level responses from the operator by providing a menu of options available at any time. This eliminates the need for the user to understand and remember a multitude of commands to select various responses from the system. For example, an application program for general inquiry concerning a group of customers might allow the operator to examine credit, past orders, or payment status by simply pressing one of the softkeys. The functions of the eight keys are dynamically labeled in a window along the right side of the screen.

During program development, the windows and softkeys are used to display various aspects of the programming environment and to accept directions from the programmer concerning the editing of statements, correction of errors, and so on. This approach can virtually eliminate the need for program listings during the development process. In addition, the concept of the language subsystem (see article, page 9) makes it possible to manage the entire set of files and software modules through a single, consistent user interface. The HP 300 is actually easier to program than many interactive interpreter-based machines, yet it provides the flexibility and efficiency of a multilingual compiler-based system.

Reliability

The HP 300 has been designed with reliability as a major requirement. This is evident from the fact that there is no periodic maintence on the basic system except for an occasional change of the fan filters. In the event that a failure does occur, an extensive set of hardware and software tools have been built in to assure quick and easy repair. Each time the system powers up, the processor runs a microcoded self-test that includes an in-depth check of the processor itself, main memory, and the input/output channel. During the same time, the fixed system disc, the flexible disc, and the IDS are performing their own self-tests, so that when the system begins operation there is high probability that the hardware and microcode for the system and its primary peripherals are functioning properly. These self-tests may also be invoked through switches located on the various circuit boards just inside the rear door or through off-line program control. The results of the tests may be viewed on a bank of LEDs for quick interpretation of any failures that occur.

System verification beyond the self-test capability may be accomplished with an extensive set of diagnostics provided for the system and all its peripherals. These programs have been structured so that unskilled users can run them by following the instructions that appear on the IDS when the stand-alone diagnostic/utility package is launched. The tests are thorough yet fast, taking advantage of features such as data loopback and remote invocation of extended self-tests to yield a group of diagnostic tools that can be routinely run by the owner to verify that the system is in good working order. Each diagnostic provides a simple go/no-go message to the operator, but can also output highly structured failure information to assist the service engineer in isolating system faults.

The HP 300 features error-correcting main memory, which corrects all single-bit errors and detects all double-

bit errors without intervention by the user. Various other errors are logged in a dedicated file on the system disc for analysis by service personnel. In addition, a system trace table maintained in main memory contains a list of the most recent system events. This provides a history of activity just before any failures that may occur, permitting the failure to be characterized and reproduced. Even a list of the most recent console commands is kept on a disc file. All these tools assure that any repair will be fast and orderly with a minimum of down-time.

Expansion

Essentially all the standard and optional peripherals for the HP 300 use a common hardware interface. This allows discs and printers to be added to the minimum system without the need to add special interface boards. In most cases the owner can purchase these peripherals and add them to the existing system without lengthy installation procedures or a service call from an HP representive. New devices are configured into the operating software using the SYSTEM BUILD utility, which leads the user step-by-step through the configuration process, taking full advantage of the IDS and its softkeys. Any errors that the operator makes in defining device type or address are automatically detected through a self-identification feature that all HP 300 peripherals contain.

The basic HP 300 System includes 256 kilobytes of error-correcting semiconductor main memory, the integrated display system, a 12-megabyte fixed disc, a one-megabyte flexible disc, and a general I/O channel, all in a self-contained package only slightly larger than a teletypewriter. Standard software includes the AMIGO/300 virtual-memory operating system, the on-line reference manual HELP, a text editor called TYPIST, a SORT/MERGE utility for file manipulations, the interactive system configurator SYSTEM BUILD, and the stand-alone diagnostic/utility package.

Available languages include HP Business BASIC and RPG-II for report generation. In addition to the AMIGO/300 file system, which provides access to files and devices, IMAGE/300 may be used for efficient management of data bases without the need for special application programs. The system can grow to include up to 16 applications terminals, two external printers, 260 megabytes of disc storage, and one megabyte of main memory.

SOS Technology

Key to achieving the capabilities of the HP 300 in such a small package is the use of silicon-on-sapphire (SOS) integrated circuit technology, with its high-speed, low-power characteristics and exceptionally high logic densities. This permits more functions to be placed on each chip, thereby optimizing performance. For example, the CPU (central processing unit) is based on a stack architecture, with the stack residing in main memory. Several of the registers that are used to manage the stack are kept in hardware (on the chips), reducing the number of memory accesses required for execution of the various stack-oriented operations. Without SOS, this would not have been possible, and the performance would have been lower. Future generations of the HP 300 will take advantage of the natural evolution of this new technology to yield even more computing power and friendliness for the business data processing needs of the 1980s.

Acknowledgments

By most means of reckoning, the HP 300 represents the most massive product development effort in Hewlett-Packard history. A number of different managers have played major roles in leading the sizable teams that made the product a reality.

Division general managers whose vision, understanding, and support were instrumental in the HP 300's development have included Dick Anderson, Paul Ely, and Ed McCracken.

Jim Cockrum, Dave Crockett, Bill Gimple, Dick Hackborn, Larry Lopp, and Tom Whitney have all provided vital leadership in capacities as lab managers through the progressive phases of the program. Dave Crockett has provided imaginative leadership as program manager and Bob Kadarauch and Jim Peachey have headed the marketing and manufacturing wings, respectively, of the HP 300 program since it assumed its present organizational structure in June of 1976.

Section managers have borne a major share of the leadership, planning, and organizational work that has helped bring the HP 300 from conception to the marketplace. In the HP 300 lab, this cast has included at various times Fred Clegg, Jim Cockrum, Bill Gimple, Jake Jacobs, Bob Jones, Jim McCullough, Peter Rosenbladt, Howard Smith, John Stedman, Phil Taylor, and the author. Bob Jones has provided for consistently good support as head of engineering services.

The lists of project managers, project leaders, and individual engineers who contributed to the HP 300 are far too lengthy for inclusion here but are largely covered in the acknowledgments elsewhere in this and next month's issue. Every past and present team member can rightfully be proud of his or her contribution to the overall achievement.

George R. Clark

Born in Florida, George Clark grew up on a farm in Indiana and received his BSEE and MSEE degrees from Purdue University in 1968 and 1973. Between degrees he picked up three years experience as an instrumentation engineer. He joined HP's Microwave Division in 1973 as an electronic tooling engineer, developing in-house test equipment for microwave semiconductors. Since 1975, he's been with the HP Computer Systems Group, involved in the design of hardware for I/O and data communications applications on the HP 300. George and his wife live in Sunnyvale and enjoy outdoor activities such as backpacking and skiing.

The Integrated Display System and Terminal Access Method

by Eric P. L. Ha and James R. Groff

THE HP 300 COMPUTER SYSTEM is designed for on-line, multiterminal applications processing. Any combination of up to 16 HP 264X and HP 2621 Terminals can be included in an HP 300 system. These terminals are dedicated to applications processing. They cannot control system operations, and they operate totally under the control of HP 300 application programs.

The application programs control the application terminals via the terminal access method software, a set of callable HP 300 system services that enable the application programs to perform input/output to terminals as if they were done to sequential files. These system services also provide simplified program control of additional terminal features such as block-mode input, display enhancements, cursor manipulations, and peripheral devices such as tape cassettes and printers.

Integrated Display System

The HP 300 system is controlled from the integrated display system (IDS) that forms the upper part of the system unit. The IDS is especially designed to provide a highly interactive, easy-to-use environment for the HP 300 user. Its advanced display and editing functions and pushbutton-oriented operation give the user direct, personal control over all aspects of HP 300 operation. The IDS also serves as a programming station for developing HP 300 application software. Application programs can also use it as an application terminal.

The IDS keyboard, Fig. 1, includes a main typewriter key group, a numeric keypad, and separate control key clusters for editing and display control. The screen displays 1920 characters in a 24-row-by-80-column format. A full 128-character upper/lower-case character set is standard, and there are optional character sets to display international and mathematical symbols, large characters, and line-drawn forms. For formatted screen displays, the IDS includes display enhancements for blinking, half-bright, underlined, and inverse video (black-on-white) fields. These can be combined with special IDS format modes for forms-oriented screen processing.

In addition to these basic features (which it shares with the HP 300 application terminals), the IDS incorporates a set of advanced display features that offer significant new display capability. Through the IDS windowing feature, the display screen can be divided into multiple sections (called windows) for greater display flexibility. Using several windows, the IDS can simultaneously display several different kinds of information on a single screen. Or, windows can be used to perform several functions at once on the IDS, with each function handled in its own separate window.

Eight softkeys bordering the right side of the IDS screen provide a pushbutton choice capability for the IDS user. The softkeys are used in a variety of ways to represent alternative actions or special functions that can be invoked at the press of a button.

The special features of the IDS are complemented by a set of terminal access method system services that further enhance the raw capabilities of the hardware. These system services are available to both HP 300 system software and application programs.

Softkeys

The IDS softkeys can be used to provide a pushbutton choice capability for the user. Each softkey can be individually labelled on the adjacent screen area with up to three lines of label information. The labels can be changed dynamically under program control to constantly indicate the function each key performs. The softkey label window can be made as wide as necessary to accommodate lengthy labels.

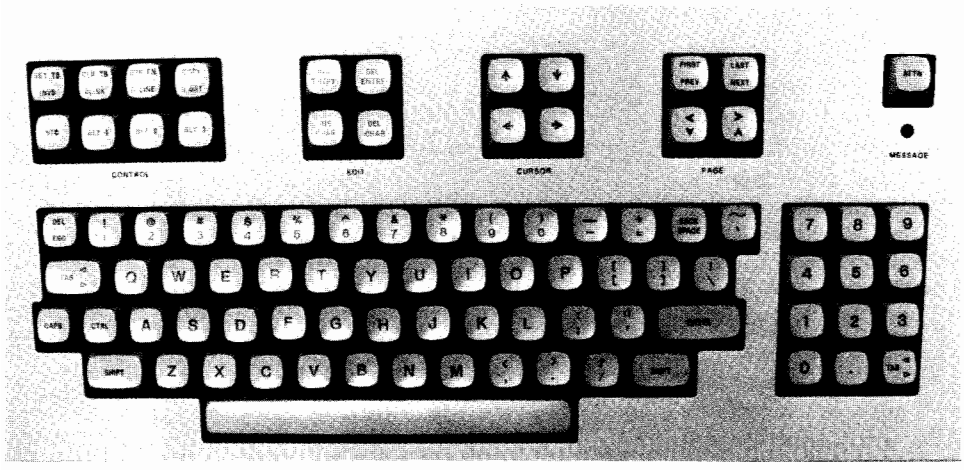


Fig. 1. *The HP 300 keyboard.*

Fig. 2. *Eight keys at the right side of the HP 300's integrated display system (IDS) can be assigned functions dynamically by application programs. The functions are displayed in a window next to these softkeys.*

Softkeys can be used for selecting one of several choices, as in Fig. 2, or they may allow multiple responses. In the latter case, the keys are set into a nonterminating mode that allows many of them to be pressed in response to a single input request. The application program can then determine which keys were pressed, and in what order. Softkeys operate totally under control of the application program, which calls system services to label them and accept softkey input.

Windows

HP 300 applications programs can use the IDS windowing capability to present information from many different sources in a straightforward way on the IDS screen. This is a frequent requirement in complex inquiry applications, such as the one illustrated in Fig. 3. In this example, a single-line window at the top of the screen identifies the displayed data as an aged accounts receivable report. The body of the report occupies the large display area in the center of the screen, and summary totals are held in a third window at the bottom. Other examples of windowing can be found in other illustrations in this article.

Windows are visually indicated by dotted-line borders on the IDS screen. These are automatically generated by the IDS between character positions and do not reduce the number of available display positions. Each window is independently controlled, and functions like a mini-display screen, with its own input capabilities. This allows even complex, dynamically changing displays to be constructed, as illustrated in Fig. 4.

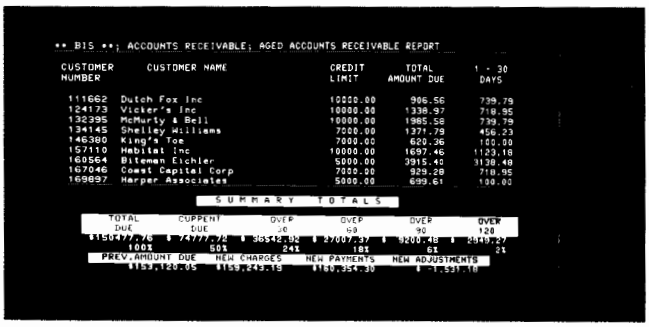


Fig. 3. *The IDS screen can be divided into independent windows, each of which acts like a separate display. Windows are indicated by dotted-line borders on the screen.*

In Fig. 4, the IDS is used as a dispatcher's screen in a shipping/receiving department. As trucks come and go at the various docks, clerks enter and retrieve data at HP 300 terminals located there. The display on the IDS constantly shows the status of each dock in the window on the left, while an inquiry and scheduling function is performed on the right side of the screen.

Up to 32 windows can be created in the IDS at any one time. Each window is either open (displayed) or closed (not displayed), with the restriction that concurrently open windows may not overlap. Output is permitted to both open and closed windows. For example, a new display can be readied in a closed window while waiting for a user response to the current display. All windowing functions—including setting window boundaries, creating, destroying, opening and closing windows, and input and output to windows—are under the control of the application program, which calls system services to manipulate the IDS.

The total amount of data contained in windows in the IDS may exceed the actual display memory size. When data is being output to windows, it is backed up in the HP 300 system by the terminal access method software. The IDS gives priority for the use of display memory to windows that are currently open. When a window is changed from a closed to an open state, its data will be replenished from the system if it has been discarded to make room for other open windows.

Viewing Data Files

Another major IDS feature is the ability to view HP 300 data files through IDS windows. Relative and keyed sequential files with type DOUBLE keys can be directly attached to a window, with the records of the file displayed in consecutive rows of the window. The window thus becomes an actual window into the file, displaying data exactly as it appears there (see Fig. 5).

Scrolling

Files that are too long to fit in an IDS window can be viewed by vertically scrolling them past the window. Using keys on the IDS keyboard, the user can move the window up and down over the attached data file a line at a time, bringing into view records above or below the displayed section of the file. Other keys move the window forward and backward a page at a time, or directly to the beginning or end of the file.

File viewing and vertical scrolling allow even lengthy

Fig. 4. *Windows in a real-time application.*

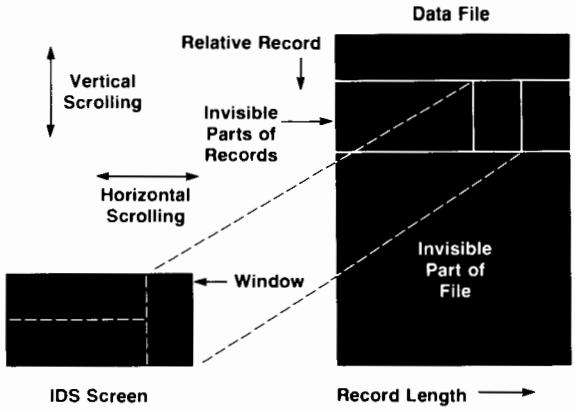


Fig. 5. *Files can be attached to windows so that data can be displayed exactly as it appears in a file. Each window may be scrolled independently through its attached file, up or down, left or right.*

files or reports to be viewed directly at the IDS. They can also be combined with more extensive application programming for sophisticated inquiry applications like the one shown in Fig. 6. In this example, a customer file is attached to the upper window for viewing. To see more detailed information on a customer, the user can scroll that customer into the inverse-video row and select the appropriate softkey, resulting in the display in the lower half of the screen.

Horizontal scrolling is used to view files that are wider than an IDS window. Using keys on the IDS keyboard, the user can move a window left and right through an attached file to directly view records up to 160 characters long. As Fig. 7 shows, horizontal scrolling is useful for presenting more information than a single screen can hold, or for previewing printed reports.

Horizontal and vertical scrolling are both available whenever a file is attached to an IDS window by an application program. The program has complete control over attaching files, enhancing window rows, selecting windows for scrolling, and so on.

Automatic File Update

Files can also be attached to windows in an automatic update mode that permits not only direct viewing, but direct modification of the data in the file. In this mode, the

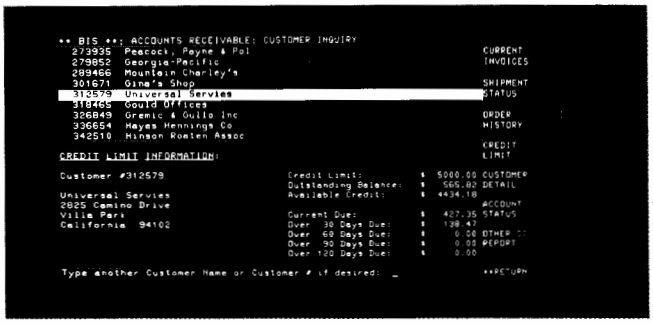


Fig. 6. *In this customer record application, the user can scroll the customer of interest into the inverse-video row and press the appropriate softkey to see detailed information on that customer displayed in the lower window.*

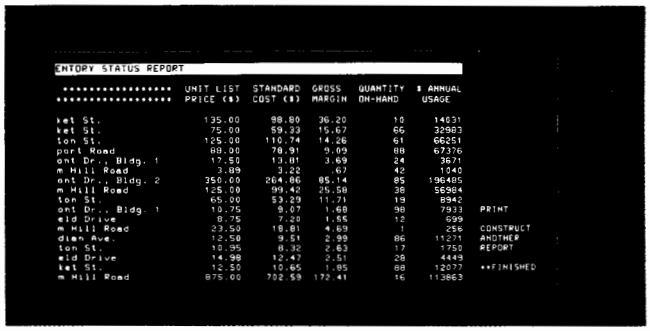


Fig. 7. *Horizontal scrolling is useful for presenting more information than a single screen can hold, or for previewing reports to be printed.*

user can respond to input requests from the application program by scrolling the file past the screen, moving the cursor about, and directly modifying the displayed data with the IDS editing keys. Terminal access method automatically updates the attached data file, record by record, as the screen image is changed. The application program is also informed of the identity and contents of each record as it is modified.

Automatic file update provides a natural editing capability that is especially useful in text processing applications, because it assures that the screen image always exactly matches the data in the attached file. To prevent unauthorized file modifications, automatic file update must be explicity invoked by the application program.

Sharing The IDS

The attention feature of the IDS is implemented by a special set of terminal access method system services that are available only to the HP 300 system software. This feature is useful for responding to special requests that come in the form of interruptions to normal processing. An application running on the IDS or a long-running command can be interrupted at any time simply by pressing the ATTENTION key on the IDS keyboard. This key returns IDS control to the operating system, making it possible to enter commands, start new programs, or use the IDS for another application as required. The attention facility can be invoked repeatedly, allowing interruptions to interruptions, as shown in Fig. 8.

When an application or command is interrupted with the ATTENTION key, it continues to execute so long as it does not attempt to use the IDS for input or output (that is, the ATTENTION key only takes away IDS "ownership"; it does not automatically suspend execution). This allows several different commands or jobs to execute in parallel. Later, the IDS can be reconnected to the interrupted command or job with no loss of data.

The attention feature can also be used to share the IDS among several different jobs that use it infrequently. This is especially useful in applications that require supervisor intervention at the IDS only in exceptional circumstances or for a short startup dialogue. When the IDS is being used and another job requests it, the system software lights the message light on the IDS keyboard to inform the user. The user can interrupt current processing (using the ATTENTION key) to determine which job is requesting the IDS, and then decide to communicate with that job or resume the

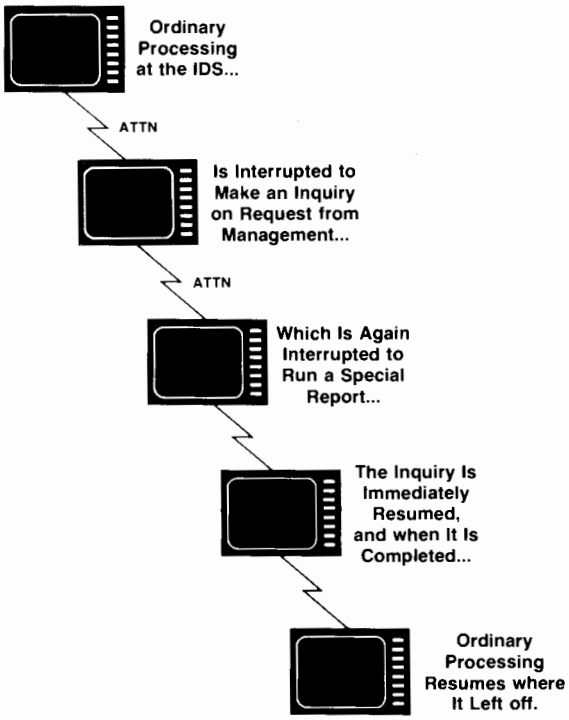


Fig. 8. *The ATTN (attention) key on the IDS returns IDS control to the operating system, making it possible to enter commands, start new programs, or use the IDS for another application.*

prior work. The HP 300 automatically manages contention among jobs for the IDS and assures that no data is lost in shifting from one job to another.

Acknowledgments

We wish to express our appreciation to Al Knoll's project team that developed the IDS: Al, Norm Marschke, Hal Sampson and Bernard Stewart designed the hardware; David Delano, Tom Gilbert and Wing Chan wrote the firmware; and Nellie Monsees did the prototype assembly. Development of the terminal access method software benefited tremendously from the participation of Bill Parrish and Mike Ard; Mike also contributed significantly to the development of the IDS firmware.

Nowait Input/Output

Input/output-intensive applications can use the "I/O without wait" feature to overlap I/O and processing for increased performance. Using nowait I/O, a program can request an I/O operation (e.g., input from an application terminal) and then proceed with other processing while the I/O operation completes (e.g., while the user types the reply). Later, if the program comes to an explicit WAIT statement before the I/O completes, it will wait for the I/O to complete before proceeding further.

Nowait I/O techniques are especially useful for multiterminal applications. Nowait I/O can also be selectively used in combination with other techniques to optimize time-critical portions of an application.

James R. Groff

Formerly a commercial applications programmer/analyst, Jim Groff joined HP and the HP 300 marketing group in 1976, and is now applications manager. A 1974 graduate of Massachusetts Institute of Technology with a BS degree in mathematics, he holds an MBA degree from Harvard University, received in 1976. Born in Palmyra, Pennsylvania, Jim is married and lives in San Jose, California. He's active in his church and enjoys reading, classical music, and travelling.

Eric P.L. Ha

Eric Ha developed the terminal access method software for the HP 300. Now a project manager at HP's General Systems Division, he's also done microprogramming for HP 21MX Computers, and before coming to HP in 1972, developed software for computer graphics and data communications. Born in Hong Kong, he received his BSEE degree in 1962 from the University of California at Berkeley and his MSEE degree in 1965 from the University of Michigan at Ann Arbor. He's married, has two children, and enjoys classical music, reading, and swimming. Eric and family live in Cupertino, California.

Reducing the Cost of Program Development

by Frederick W. Clegg

IN TYPICAL SCENARIOS of twenty years ago, virtually all production program development was done using unit record equipment to prepare a source tape or card deck that was then submitted for processing with a batch of other jobs. Today, by contrast, most program development is done in an environment characterized by a greater or lesser degree of dialog between the programmer and the computer through an on-line terminal. To the user of some of today's more pleasant program development facilities, the computer appears to understand the language being

used; a number of currently available implementations of BASIC and APL, for instance, convey this impression quite thoroughly. Of course, virtually no machines directly understand BASIC, APL, or any other high-level programming language. Almost invariably, the description of some sequence of computing operations written by a programmer to tell a computer how to perform some desired task is translated into one or more intermediate representations before the computer can carry out the programmer's instructions. The translation is done primarily to make the most efficient use of both the computer's storage capacity and its processing capabilities. However, the gains in efficiency do not come free. The translation inevitably consumes some of the computer's resources. More important, a significant toll is often extracted in terms of the time spent and the distractions endured by the programmer.

To gain a fuller understanding of the price associated with using translations and intermediate representations to achieve greater program efficiency, let us consider briefly a typical program development scenario, as summarized in Fig. 1. Once a programmer has completed the writing of a program or a program module (i.e., having written the source code, be it on a coding form or the back of an envelope), the next task is to get this source code into the computer. With most modern systems, this is done through the use of an editing program—EDIT/3000 being a good example. The programmer converses with this editor typically through an on-line terminal such as an HP 2645. The editor allows source text typed by the programmer to be stored in magnetic disc and/or tape files and permits convenient correction of typing and logic errors. To get even this far, the programmer needs to know not only the language in which the source program has been written, but also how to access the editor and at least a subset of the command language used to invoke the various functions of the editor. Just getting this far often takes even an experienced programmer several full days when confronted with an unfamiliar computer for the first time.

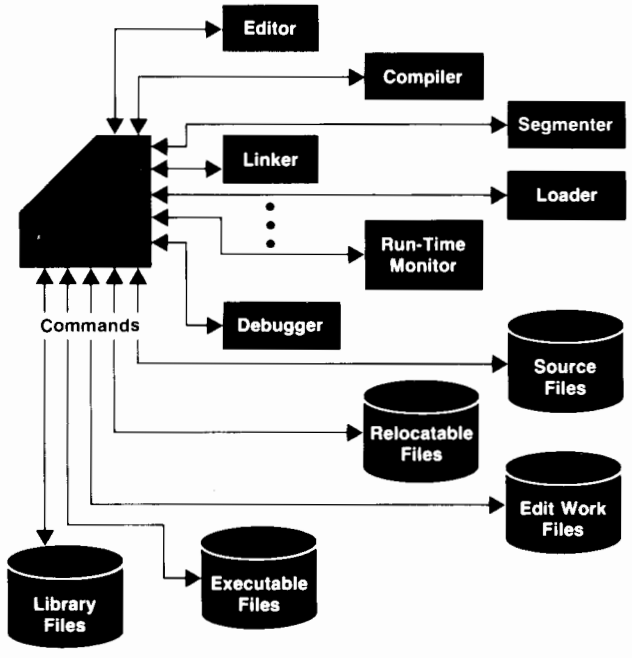


Fig. 1. *A typical program development scenario.*

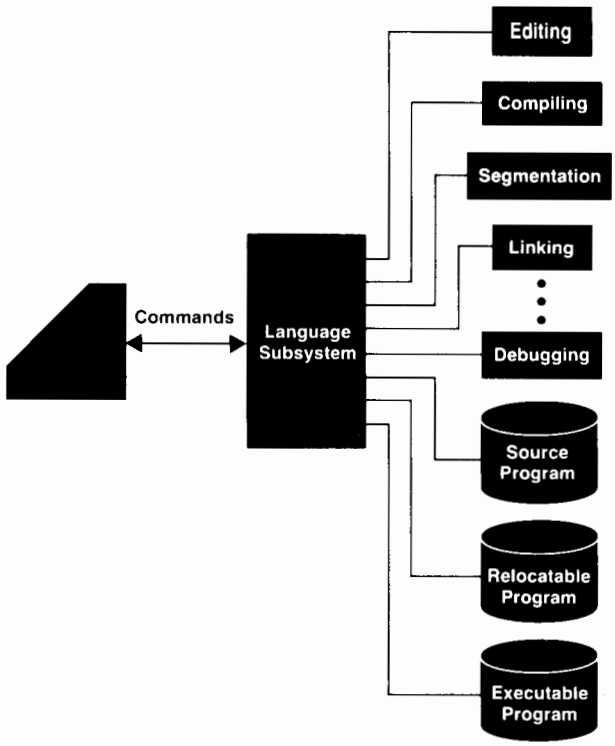


Fig. 2. *HP 300 program development is simplified by the language subsystem concept.*

Once the source code for a program has been entered into one or more files (called, appropriately, source files) on the computer, the first translation to an intermediate representation is undertaken. This translation is accomplished by a program usually called a compiler, which reads the source code file(s) and produces an output in a form similar to the machine code the computer can directly execute. Compiler output is usually modular as defined by the boundaries of various program units such as procedures, subroutines, code segments, or whatever. These output modules typically contain references among themselves and to various capabilities in the computer's operating system software. Such references, more often than not, are not resolved by the compiler, but rather after compilation is complete are resolved by some separate program called a linkage editor or (as is the case on the HP 3000) a segmenter. The output of this last program might best be characterized as an almost-ready-to-run version of the user's program. Even at this point, however, some additional services such as final linking, loading into memory, and the like are commonly required to bring the user's program into control of the computer.

These various translation steps and associated intermediate forms permit more efficient and flexible use of computer resources by the final running version of the program than would otherwise be possible. The price paid for this efficiency is imposed upon the programmer and/or the company paying the programmer's salary. The programmer needs to know not just the programming language in which the original program is written, but also typically

three to six totally different other languages—the command languages required to operate the editor, compiler, segmenter, loader, and so on. Indeed, it is the learning of these minor details that usually constitutes the largest hurdle to be overcome by a person learning to use a new computer.

The software designers of the HP 300 believe that the need for familiarity with the numerous translation steps and intermediate representations required to get a production-level program running on a typical computer is a major contributing factor in the oft-bemoaned high cost of software development in today's computing industry. This concern led us to depart from tradition in significant ways and to conceive for the HP 300 a highly integrated program development environment that differs in a number of dramatic ways from the typical scenario painted above.

HP 300 Language Subsystem Concept

From a very early point in the design cycle of the HP 300, it was decided that the application programmer should perceive a single, integrated environment throughout the program development cycle. That is, the programmer should not be required to invoke an editor, work on the source program, exit the editor, invoke a compiler, do the compilation, exit the compiler, invoke the segmenter, and so on. Instead, we wanted the HP 300 programmer to be able to complete all phases of software development, from initial source entry to completion of final testing, from the same place in the system. Now, this goal is rather easily achieved through the use of a fully interpretive language processing system, such as the versions of BASIC found on numerous computer systems (and even the HP 3000's BASIC interpreter). It is characteristic of such interpretive systems, however, that they consume considerably more storage space and execute much more slowly than comparable programs that have been subjected to the several translation phases sketched above. Primarily in the interest of run-time performance, it was decided that all of the HP 300's program languages would be supported by compilers. This decision notwithstanding, the goal of a highly integrated, convenient program development environment was not to be compromised. In short, then, our objective was to obtain the execution-time performance characteristics of compiler-based systems and at the same time to provide the convenience and productivity-improving features found heretofore only in interpreter-based program development systems.

To obtain the performance, power, and flexibility inherent in compiler-based program development systems, most or all of the system programs discussed in our earlier scenario (i.e., editor, compiler, segmenter, etc.) play essential roles. Indeed, each programming language subsystem on the HP 300 incorporates exactly such components. Our goal, however, was to make knowledge of how to get to and operate each of these components—indeed, knowledge of their very presence—unnecessary for the development of programs on the HP 300. Our implementation to achieve this goal is shown symbolically in Fig. 2. A central role is played by a component called a language monitor. This is the central, supervisory program in a language subsystem. It is analogous to an automatic switchboard that determines when the programmer is ready for the services of an editor, compiler, segmenter, or the like, and then automatically invokes the services of that component, without explicit direction from the user to do so.

Closely related to the language monitor is a second key concept in each HP 300 programming language subsystem: the program workspace. Simply defined, a workspace is a collection of all files related to the development of a given program. Instead of keeping separate, explicit track of all the source, relocatable, and other files associated with a program, the HP 300 programmer needs to remember only a single program name.

The detailed organization of an HP 300 program workspace is shown in Fig. 3. The root file serves as a directory to the rest of the workspace. The prolog file contains parametric information such as IDS (integrated display system) tab stops, parameters (e.g., line number increments) used for editing, file equations to be used at execution, and the like. The source text entered by the user resides in one or more source module files. Initially, a workspace always has a source module named MAIN, and other modules may be created and given arbitrary names by the user.

The compiler output files (header file and relocatable file) and the debug file are all written to by the compiler. Of particular interest is the debug file, which permits fully symbolic debugging when a program malfunctions. To inspect the value of a variable called Rate immediately before execution of statement 230 in a BASIC program, for instance, a programmer needs no information whatsoever about the numerical memory addresses occupied by the object code corresponding to source statement 230 or the address associated with the data item Rate. The programmer need only tell the symbolic debug facility (in the BASIC language subsystem in this case), BREAK AT 230, and then, when the resulting breakpoint is encountered, PRINT Rate.

The segment file is produced by the HP 300 segmenter as a result of processing the header and relocatable files during program preparation. At the same time, the segmenter updates certain information in the debug file to account for intersegment references.

After successful program preparation, the segment file is read by the linker, which resolves references by the user's program to services in the AMIGO/300 operating system and libraries, and produces the contents of the execution file, essentially the execution-time, memory-image, object code representation of the user's program, ready to be loaded and executed without further processing.

Each time a user program is created or altered on an HP 300, the language monitor automatically invokes the services appropriate to what the user is doing at any given point. Each component providing these services acquires access to the appropriate representation of the user's programs (i.e., to the appropriate files associated with that program) through the workspace management modules of the AMIGO/300 operating system.

Writing a Program on the HP 300

Now let us turn to a scenario of a typical program development session on the HP 300 in an attempt to capture, as much as is possible on these pages, the personality of the HP 300 as perceived by the programmer. Most of the examples will be taken from the BASIC subsystem, since this is the most widely known of all the languages currently sup-

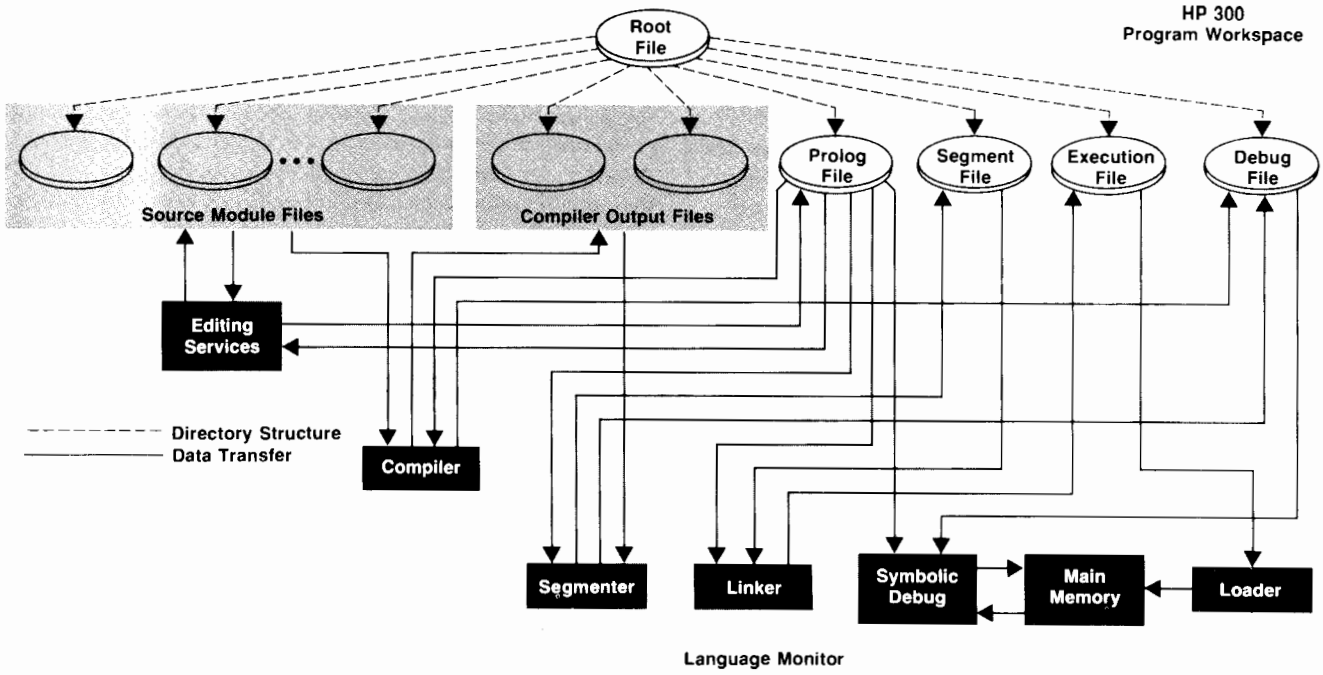


Fig. 3. *Organization of an HP 300 program workspace.*

ported on the HP 300. Later we will explore certain features of the RPG-II subsystem that represent significant innovations to the RPG programmer.

Access to a programming language subsystem on the HP 300 is achieved simply by typing the name of the desired language into the command window of the integrated display system (IDS) while in the AMIGO/300 operating system environment. Upon invocation, a language subsystem begins by prompting for the name of the user's program (this is in fact the name of the user's program workspace). If no workspace with the specified name is found, a new one with this name is automatically created.

Use of the IDS during program development is typified by Fig. 4. As seen here, the top of the screen is used by the one-line environment window, which identifies that the user in this case is in the BASIC subsystem, working on the module MAIN of a program called PAYROLL. Down the right-hand side of the screen is the softkey label window.

The top softkey is labelled HELP, as is nearly always the case in HP 300 system software. Any time the user is uncertain about what to do, the HELP subsystem may immediately be accessed by pressing this key. HELP is the HP 300's on-line quick-reference guide. Access to items in HELP is greatly facilitated through the use of an internal KWIC (key word in context) index and a clever inquiry analysis algorithm.

HP 300 language subsystems operate in two modes during editing operations: command mode and compose mode. In the former, the next-to-last row of the IDS screen is used as the command window, as shown in Fig. 5, in much the same fashion as in the AMIGO/300 operating system environment. Here the user specifies the operation to be performed using a single command language to access all components' services (i.e., those of the editor, compiler, segmenter, etc.) with a syntax very close to that of colloquial English. The valid command verbs accepted by HP 300 language subsystems are shown in Table 1.

Table 1
Language Subsystem Command Verbs

ADD	FIND
APPEND	LINK
BIND	MOVE
CHANGE	OPEN
CLEAR	PREPARE
CLOSE	PRINT
COMPILE	PURGE
COPY	RECALL
CREATE	RENUMBER
DELETE	RESEQUENCE
DEVELOP	RESET
DUPLICATE	SAVE
EDIT	SET
EQUATE	SHOW
EXIT	TEST
	VIEW

The bottom line of the IDS screen is always reserved as the error window and is used to report all difficulties encountered. HP 300 error messages are in English and designed to be self-explanatory rather than to require a manual to interpret. The user may switch back and forth between command and compose mode by striking the COMMAND/COMPOSE softkey at the bottom of the softkey array. One of the words COMMAND or COMPOSE is always highlighted when this key is active. The highlighted word identifies the mode the language subsystem is in.

In compose mode, language subsystems use the powerful editing features of the IDS to permit the user to enter and

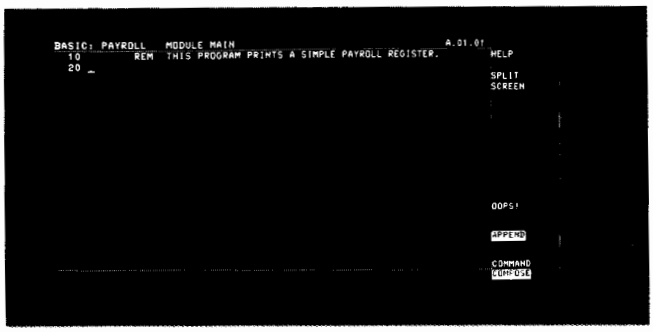


Fig. 4. *BASIC language subsystem prompting for a new line of text illustrates typical use of the integrated display system (IDS) for program editing.*

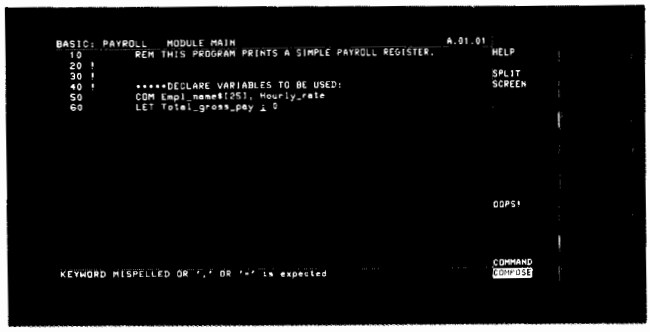


Fig. 6. *Making immediate editing changes in compose mode as indicated by the syntax checker.*

alter source text in the workspace's source module files. In this mode, the IDS cursor may be positioned anywhere in what is called the display window to make changes and additions to source text, as is illustrated in Fig. 6. In compose mode, the command window does not appear on the IDS screen. Instead, the display window is expanded downward one row to take advantage of this extra space.

The compose mode of a language subsystem is implemented through three different internal states of the language monitor: edit state, insert state, and append state. Edit state is entered to modify existing source text lines in the user's workspace. When the user specifies the name of an existing, non-empty workspace when initially prompted for the name of the program, the language monitor automatically changes to the edit state (and also, therefore, compose mode). The insert state of the language monitor is used to insert new source text between existing lines of text. A transition to this state is occasioned by striking the INS ENTRY key on the IDS keyboard, or by typing some command requiring the addition of new text between two already existing lines. That the language monitor is in insert state may be readily identified by the presence of the cursor in a space between two existing lines in tbe IDS display window. In the case of the BASIC language subsystem, an automatically generated statement number is placed just before the cursor whenever the user is prompted for a new source statement. This is true in both the append state (discussed below and illustrated in Fig. 4) and the insert state. Insert state is exited when the user strikes the INS ENTRY key a second time, performs any editing operation directed at existing text, or begins to add new text to the end

Fig. 5. *The IDS screen while the BASIC language subsystem is in command mode.*

of the present text module. In the latter case, the language monitor changes to its append state. The append state is also entered if the user strikes the APPEND softkey while in edit or insert state. While in the append state, the APPEND softkey label is highlighted in inverse video and the user is prompted by the IDS cursor (and in BASIC by automatically generated statement numbers) for additional text at the end of the present workspace source module. Append state is exited when the user strikes the highlighted APPEND softkey or requests any other operation other than the adding of text at the end of the source module.

Whenever source text is being entered or modified in an HP 300 language subsystem, whether in command mode or compose mode, it is automatically checked for correct syntax. Should an error be detected, the error is immediately explained in English in the error window and the cursor is placed in the display window at the point where the error was detected, as illustrated in Fig. 6. This powerful feature ensures that most programming errors will be detected and corrected immediately, well before any compilations are attempted.

One of the softkeys, labeled SPLIT SCREEN, causes the large area occupying the central rows of the IDS screen to be split into two smaller windows, in the manner shown in Fig. 7. The lower of these is simply a shrunken version of the display window, in which the cursor appears and editing operations are performed while in compose mode. The upper half of the screen (minus the environment window at the top) is now devoted to what is called the view window. This is a read-only window (i.e., the cursor never appears in it) that may be used to view such diverse things as parametric data in the active workspace's prolog file, source text in any module of any workspace to which the user has legitimate access, listings and maps produced by the compiler, segmenter, and linker, and the like. If the screen is split through the use of the SPLIT SCREEN softkey (rather than through the use of a VIEW command issued in command mode that explicitly indicates what the user wants to view), the default information sbown in the view window is the source text in the module the user is currently working on. To use the terminology of the HP 300's terminal access method, we describe this situation by saying that the text in this source module file (i.e., the active module) is attached to the view window. Once a given file is attached to the view window, it remains attached to that window across subsequent unsplitting/splitting operations until some later command dictates that the user wishes to view something

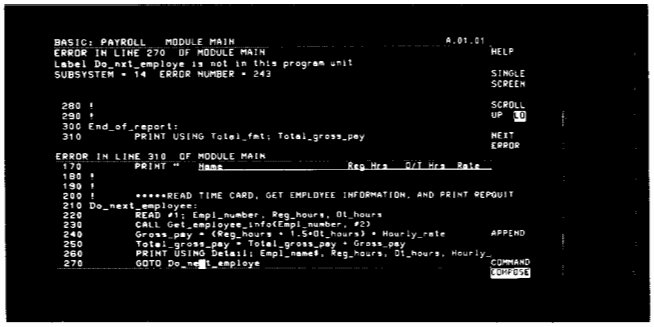


Fig. 7. *Post-compilation error correction using the split-screen feature and the* NEXT ERROR *softkey.*

else. While this split screen feature is being used, the SCROLL UP/LO softkey is active, as may be seen in Fig. 7. There is only one set of scrolling control keys on the IDS keyboard. These keys operate on either the display window or the view window, as specified by the corresponding highlighted word in the SCROLL UP/LO softkey label. While using split screen operation, the softkey formerly labeled SPLIT SCREEN is relabeled SINGLE SCREEN. Depression of this key causes the view window to be closed and the display window to be again expanded to occupy the space on the screen thus freed. When the user thus reverts to single-screen operation, the SINGLE SCREEN softkey is relabeled SPLIT SCREEN and the effects of the scrolling keys automatically revert to the display window.

Once the source text for a program has been entered and edited to the point where it is believed to be correct, the programmer ordinarily strikes the TEST* softkey. This action causes the language subsystem to perform the correct sequence of steps to bring the user's program into test execution. In the most common case, this sequence consists of compilation, segmentation, linking, and launching into execution under the auspices of the symbolic debug facility. Unnecessary steps in this sequence (e.g., compilation in the event that the source code has not been altered since the last compilation) are automatically bypassed when the TEST softkey (or the synonymous TEST command) is serviced.

Some errors a programmer can make are undetectable at the time source statements are being entered. An example of this situation is inclusion of the statement GOTO 150 in a program containing no statement 150. When errors like this are detected during compilation, another novel feature of the HP 300 language subsystems becomes apparent. If errors are detected, the automatic sequence entered in servicing the TEST softkey or command (assuming this is why the compiler was invoked) is halted, the IDS screen is automatically split, the compiler's error summary is shown in the view window, and the NEXT ERROR softkey becomes active. Each activation of the NEXT ERROR softkey causes the detailed message explaining the next error to be shown in the view window, beginning with the first error the compiler found and proceeding in the order in which the compiler found subsequent errors. Simultaneously, the actual source text in which the error was detected is made visible in the display window and the IDS's cursor is positioned within the offending statement to expedite correction of the mistake, as shown in Fig. 7. The language subsystem is in the edit state at this point, and all the editing features discussed earlier are available to aid the user in correcting the source text while viewing the errors in the other half of the split screen.

Once compilation, segmentation, and linking have been successfully achieved,* the user's program may be launched into test execution. Within the language subsystem environment, this is always done under the watchful eye of the HP 300's symbolic debug facility. When this is the case, run-time exception conditions (e.g., division by zero, array bounds violation, etc.) cause the user's program to be suspended with control transferred to the debug facility. The debug facility conducts a dialog with the user, explains what has gone wrong, and allows the user to inspect and alter the values of variables in the program, insert breakpoints (temporary, permanent, or counting), perform miscellaneous calculations in any mixture of decimal, octal, and hexadecimal bases, and, in general, figure out what has gone awry and why. A typical IDS display in the debug environment is shown in Fig. 8. It is noteworthy that all references to variable data and code locations within a user's program are ordinarily entirely symbolic. Thus the programmer need not be bothered by having to learn to decipher and to sift through compiler variable maps and segmenter/linker load maps to determine the actual numeric memory addresses of data and program code under scrutiny.

Once the programmer using the symbolic debug facility understands any anomalies and is ready to try appropriate alterations to the program, he or she may exercise the CANCEL option (either by command or softkey). This terminates the program being debugged and immediately returns the user to the compose mode of the subsystem to accommodate further editing. Thus we have completed the cycle that typifies software development efforts, from original source entry through debugging. The single most important thing to be noted in the above description of this cycle is that the programmer works the entire time from one place, the appropriate language subsystem, and does not need to switch

*An alternative label for this key might be RUN. However, a program is launched into execution through a substantially different mechanism from within a language subsystem than that used in response to the RUN command honored directly by the AMIGO/300 operating system. When a program is RUN, for instance, the services of the symbolic debug facility are not available. To avoid possible confusion arising from these internal differences, the word RUN was deemed inappropriate for this softkey label.

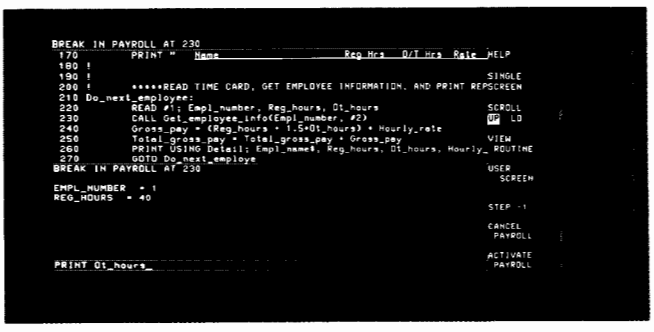


Fig. 8. *Using the symbolic debug facility.*

*Errors during the segmentation and linking phases of program development on the HP 300 are very rare except when the user is employing his or her own run-time libraries. For this reason, post-segmentation and post-linking error resolution will not be covered in this article. However, they are straightforward.

explicitly between editor, compiler, segmenter, and so on.

Novel RPG Programmer Aids

All of the examples used thus far to depict the HP 300 program development environment have been taken from the BASIC language subsystem. Much of the flavor perceived in these examples may be found in other HP 300 subsystems, as well. Many of the features of TYPIST, the text-editing package of the HP 300, directly parallel the source code entry and editing capabilities already discussed. Of course, some other features are not appropriate—the TEST function described earlier, for instance, has no meaning in the context of general document editing. While all HP 300 subsystems have many features in common, one in particular, the RPG-II language subsystem, has so many innovative features that it warrants special mention. RPG is a comparatively specialized business data processing language. One of its disadvantages in the eyes of many programmers is that it enforces a complex, position-sensitive input format—that is, specific codes for specific functions must be in specific columns of specific classes of source records. For this reason, it is impractical to attempt the writing of an RPG program for most computers without using a standard coding form. The HP 300 RPG-II language subsystem obviates this need by providing a template facility for each of the seven different source record types making up this language. RPG-II/300 is described in the article on page 20.

Conclusion

All of the features described above are highly integrated in the HP 300's language subsystem. An objective of great importance in their selection and organization was that a programmer new to the HP 300 but knowing one of the languages it supports should be able to write and debug even production-level programs after only a very brief orientation period with the machine. Features were designed to be as self-explanatory as possible. Although it is true that the programmer must learn the subsystem's command language to exploit the full power and flexibility inherent in these subsystems, every effort was made to make this command language as natural and close to colloquial English as possible, subject to constraints imposed by the automatic parser generation techniques employed. The inclusion of synonyms in the command language and the addition of a spelling-corrector/abbreviation-acceptor algorithm to the front end of the command language interpreter further enhance the friendliness and convenience experienced by the user. And of course, when the user does get stuck despite these features, there is always HELP!

One of the most troublesome and challenging of all the problems facing the disciplines of computer science and data processing today is that of the rapidly rising costs of software development. The fundamental goal of the conceivers, designers, and implementers of the HP 300's language subsystems has been to make a major contribution toward alleviating this problem by enhancing programmer productivity. Judging from initial reactions obtained from field personnel, early customers, and other new users of first-release HP 300 systems, a large measure of success has been achieved in accomplishing this objective.

Acknowledgments

Many people have been instrumental in the conception and evolution of the HP 300 language subsystems. Proposals by Denise Pitsch and Fred White had significant influence on the original design efforts. Jordan Brodsky and Mary Berner wrote the original versions of much important support code for the subsystems. Credit for the novel features of the RPG-II subsystem goes to Larry Chapin, Tu-Ting Cheng, Jon Kelley, Wendy Peikes, and Ken Van Aalsburg. The authors of the HP 300's compilers, notably May Kovalick and Denise Pitsch of the BASIC team, made important contributions to the definition of the language-monitor/compiler interfaces. Peter Schorer and Carol Fuquay conceived and did much of the early work on the HELP facility. James Miller deserves credit for the symbolic debug facility from its conception all the way through its release in HP 300 subsystems. Last, but certainly not least, the final phases of the implementation of the subsystems were accomplished by the assiduous efforts of Don Coleman, Harry Muttart, Ollie Polk, Dick Somrak, and D.D. Roberts.

Some important HP 300 user subsystems software is not explicitly mentioned in this article. Paramount here is the systems programming language designed specifically for implementation of all HP 300 systems software. Bill Barrett, John Couch, Danny Low, Rick Meyers, and D.D. Roberts were instrumental in the definition, implementation and support of this language. Danny Low and Leon Leong designed and implemented the SORT/MERGE package. Ed Dufour, Mike Lipsie, Dave Stallmo and John Trimble engineered the I/O formatter. Karen Chez, Tom Peters and Peter Lau developed the multiterminal data entry package discussed in the article on RPG-II later in this issue. Carol Chan, Judy Guist, Sue Meloy, and Fred White implemented the mathematical functions library.

The author would like to express his profound gratitude to these persons as well as the numerous others who assisted in the development of the HP 300 user subsystems.

Frederick W. Clegg

Fred Clegg is section manager for HP 300 user subsystems. Joining HP in 1975 as a development engineer, he designed the HP 300 language monitors, then became project manager for the language monitors, BASIC, SPL-II, and TYPIST. After receiving his BS degree in engineering science from Oakland University in 1965, he spent a year at Technische Hochschule Darmstadt, then attended Stanford University, receiving his MS and PhD degrees in electrical engineering in 1967 and 1970. He served as instructor of electrical engineering at Stanford from 1969 to 1970 and was assistant professor of electrical engineering and computer science at the University of Santa Clara from 1970 to 1975. He's a member of IEEE and the IEEE Computer Society, and is listed in *American Men and Women of Science*. Born in Atlanta, Georgia, Fred is single, has a daughter, and lives in San Jose, California. His interests include aerobatic flying, trap shooting, amateur radio, and electronic tinkering.

Managing Data: HP 300 Files and Data Bases

by Phillip N. Taylor, Alan T. Paré, and James R. Groff

DATA IN THE HP 300 SYSTEM can be organized in two ways: files and data bases. The software to manage these capabilities consists of the AMIGO/300 file system and IMAGE/300, respectively.

File System

The AMIGO/300 file system manages HP 300 data storage and controls access to all HP 300 files and devices. The file system automatically handles all low-level file management tasks such as disc management, buffering, blocking, device handling and device allocation. It offers the programmer a broad set of high level, device-independent capabilities for data storage and access.

A subset of the AMIGO/300 command language is used to manage files and devices from the integrated display system. Program access is provided by the input/output structures of the HP 300 programming languages, or through callable HP 300 system services. In addition to these on-line capabilities, the HP 300 diagnostic and utility system provides a stand-alone environment for disc formatting, system volume restoration, and other off-line functions.

File System Organization

In the HP 300 file system, a **file** is a named collection of records, such as a file of timecards or accounting transactions. A **file domain** is a collection of files grouped together for reference, such as all the files for a given application. File domains can be protected with passwords to prevent unauthorized access. A **device** is a physical unit for data storage, input, or output, such as a disc drive, terminal or printer. A **volume** is a piece of physical storage media (such as a flexible disc) that resides on a storage device.

Every file, domain, device, and volume on an HP 300 system is identified by its user-assigned name. These names can be combined into a fully qualified filename that uniquely identifies a file or device. Fig. 1 shows an example of a fully qualified filename and its interpretation.

For non-file-structured devices (such as printers or terminals), only the device portion of the fully qualified filename is needed (e.g., .PRINTER or .TERMNL4). In practice, fully qualified filenames are almost never used. Instead, the system assumes default values for omitted parts of the name, so files can generally be referenced with simple names such as CHECKS or ORDERS.

As Fig. 1 shows, the HP 300 has a unified naming scheme for files and devices. There is also a uniform set of procedures for accessing both files and devices, and a uniform set of commands for file and device management. The result is a high level of file and device independence that simplifies both application design and system operation.

File and Device Access

The file system provides both serial and keyed file access. In serial access, records are processed in order, in a forward or backward direction. Serial access is used to process a file in sequence and to access sequential devices such as printers and terminals. In keyed access, records are accessed randomly, based on a key associated with each record. Keyed access is for selective retrieval of records from a file. Both access methods can be used to input, output, add, delete, and replace records as needed, limited only by the characteristics of the file or device being used.

The file system offers a choice of seven different file structures for storing data. Each structure meets a particular application need, such as rapid random access or simple sequential retrieval.

Sequential files provide rapid serial access to the records in the file. Records are stored in chronological order as they are entered, and are also accessed serially. Sequential files are efficient for data that is always processed in a fixed sequence, such as transaction files.

Relative files provide access to records based upon their relative record number within the file. Records are stored in order by relative record number and access is either serial or random by relative record number. Relative files are often used to store data that is accessed through pointers in other files.

Keyed sequential files provide access based upon a record key associated with each record. Records are stored in order by key, together with a key index that is used to access them. Access is serial in record key order or random by record key. Keyed sequential files are used to store data that must be processed both randomly and serially in key order, such as a customer master file.

Direct files provide rapid access based on a record key associated with each record. Records are stored by applying a hashing algorithm to the record keys, which tends to distribute the records evenly through the file. Access is random by record key, or serial in physical storage (not record key) order. Direct files are used for data that requires rapid keyed access and little or no serial processing.

Library files allow collections of logically related files to

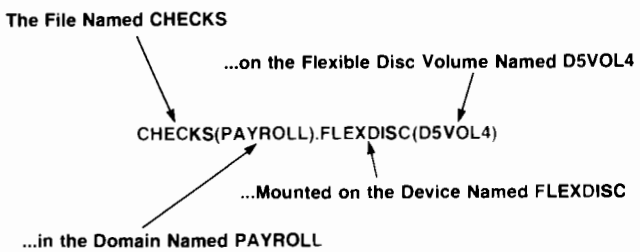


Fig. 1. *A fully qualified HP 300 filename. In general, most parts of the name can be omitted and the system will assume default values.*

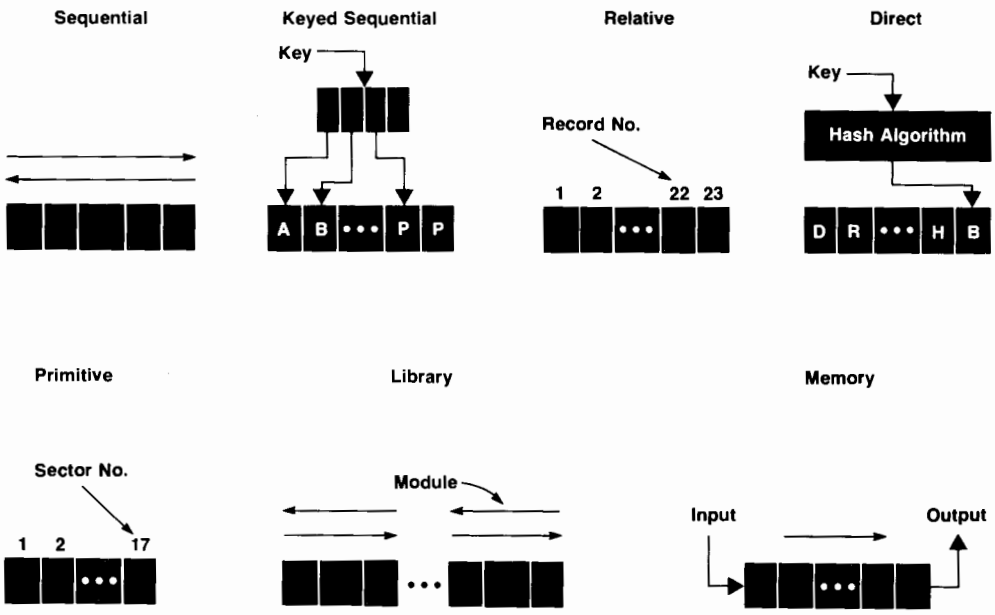


Fig. 2. *The HP 300 file system offers a choice of seven file structures for storing data.*

be stored together in one common file (a library). A library file is composed of one or more named modules that are accessed randomly by name. Each individual module has the storage and access characteristics of a sequential file. Library files are generally used to store collections of things, such as a set of terminal screen formats.

Primitive files provide low-level file access for sophisticated application programmers. In a primitive file, records correspond to logical disc sectors within the file. Access is random by relative record number (i.e., sector number) and data is transferred in sector multiples. Primitive files are especially useful for programmers who want to augment the HP 300 file structures with a customized structure.

Memory files provide efficient program-to-program communications between one or more sending programs and a receiving program. Records are stored in a circular first-in-first-out buffer in virtual memory. Access is serial for both writing data into the file and reading data from the file. Memory files are used for communication and synchronization among multiple programs or tasks in an application system.

In addition to these seven file structures, the file management system offers the following capabilities for storing and accessing data:

- Variable length records. Records may vary from one to over 2,000 characters in length for efficient disc space use. The file system automatically manages and recovers space when records are replaced or deleted.
- Domain security. Passwords on private file domains prevent unauthorized entry into the domains.
- Dynamic file allocation. Storage space is automatically allocated for data files as they need it.
- Private volumes. Removable private disc volumes can be used to transport files to other HP 300 systems or to share a single storage device among several sets of data.
- File sharing. Programs can obtain exclusive access to a file, or multiple programs can concurrently access the file in a read-only sharing mode or an update sharing mode (with file locking and unlocking).
- File equation. HP 300 programs can specify the files and devices they use as logical file names. These logical names are associated with actual physical files or devices through file equations that can be changed independently of the program. File equations can be stored with the program as default file assignments, and they can be entered as commands at execution time.
- Input/output without wait. Using nowait input/output, a program can initiate an I/O operation (such as input from a terminal) and then continue with other processing before the I/O is complete (e.g., before the user types in the response). Sophisticated programmers can use this technique to increase performance in I/O-intensive applications by overlapping I/O and processing.

IMAGE/300 Data Base Management System

IMAGE/300 is the data base management system for the HP 300 computer system. IMAGE offers an alternative to conventional file systems, and can help to reduce data redundancy and promote consistency, timeliness, and integrity of data, allowing it to be more responsive to user needs. Using IMAGE, the data for an entire application can be stored in an integrated, highly structured data base. Application programs use this structure to access the data and derive information about relationships among the data items. In addition, the IMAGE data base inquiry facility allows the user to access and inquire into the data base without application programming.

IMAGE/300 consists of several components:

- A *schema processor* that translates a data base schema (a formal data base description written in the IMAGE data base definition language) into an internal data base representation
- *Commands* for creating, purging, erasing, storing and restoring the data base
- *System services* that are used by applications programs to access the data base
- A *data base inquiry facility* for making impromptu data base inquiries and for data base testing and debugging.

IMAGE/300 Data Base Structure

An IMAGE data base consists of data items, data entries, and data sets. A **data item** is a single piece of data, such as an employee number or employee name. A **data entry** is an ordered set of related data items, such as all the information about a particular employee. A **data set** is a named collection of data entries, such as the set of all information on all employees. A **data base** is a named collection of related data sets, such as all the data sets that relate to a payroll application.

Fig. 3 shows how data items, data entries, and data sets relate to one another, using a payroll application as an example. As shown in Fig. 3, data entries are related by two different types of IMAGE data sets:

- *Master data sets* are used to store data entries that represent uniquely identifiable entities. The storage location assigned to each master entry is determined by the value of a specific data item within the entry. This data item, called a search item or key, serves as the primary identification for that entry. All entries in a master data set have the same search item, and each entry has a different value for that item. In the example of Fig. 3, EMPLOYEE and PAY-PERIOD are master data sets. EMPLOYEE NUMBER and PAY DATE are their respective search items.
- *Detail data sets* are used to store entries that represent related entities. In a detail data set, the storage location assigned to a particular entry has no relation to its data content. When a new entry is added, it is placed in the first available location. Unlike master data sets, a detail data set may have up to 16 search items, and the values of a particular search item need not be different for different entries. In general, many entries will have the same value for a given search item.

An important purpose of master data sets is to serve as indexes to detail data sets. Data entries in a master data set contain pointers to groups of entries in detail data sets that have the same search item as the master set. In Fig. 3, TIMECARDS is a detail data set that is indexed by both the EMPLOYEE and PAY-PERIOD data sets. A master data set may be related in this way to more than one detail data set, and a detail data set may be related to more than one master data set.

To represent data relationships, master and detail data sets are combined in a network of data sets that forms an entire data base. This network not only stores data, but represents relationships among pieces of data as well. The data can then be retrieved based on these relationships.

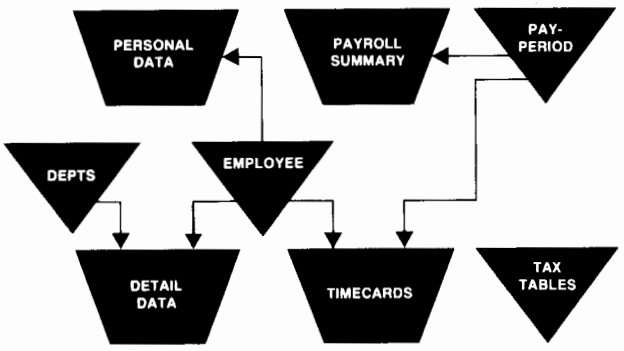


Fig. 3. *An IMAGE/300 data base consists of a network of related master data sets (EMPLOYEE, PAY-PERIOD) and detail data sets (TIMECARDS).*

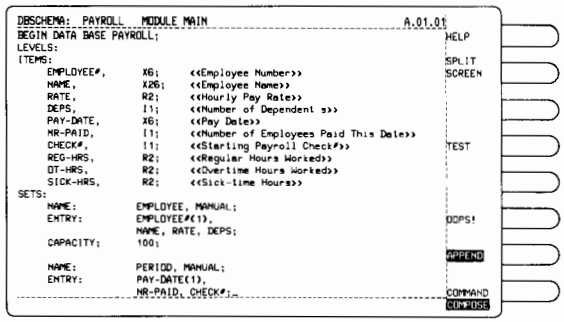


Fig. 4. *Entering the schema that describes the structure of the data base of Fig. 3.*

Creating an IMAGE/300 Data Base

To create an IMAGE data base, the user must first decribe the data base structure to the HP 300 system. The description is called a schema, and it defines the data items, data entries, and data sets that make up the data base, as well as other capacity and security information. The IMAGE schema processor, DBSCHEMA, is used to enter the schema, as illustrated in Fig. 4. The partial schema shown corresponds to the payroll data base example in Fig. 3.

The schema processor creates an interactive environment for entering and editing schemas that closely resembles the TYPIST text editing environment. When the schema has been entered, the TEST softkey compiles it into an internal form and reports any errors for immediate correction. After the schema has been defined, AMIGO/300 commands can be used to create, purge, erase, store and restore the data base.

IMAGE/300 Data Base Access

Application programs access IMAGE data bases through a set of IMAGE system services. These services give the programmer high-level, applications-oriented data base access, without concern for where the data is stored or how it is accessed. Services are available to open and close a data base for access, obtain information about the data base, read all or some of the data items in a specific data entry, add a new data entry to the data set, update data item values in an existing data entry, delete a data entry from the data set, and lock and unlock the data base or a subset of it for temporary exclusive access.

Using these services, programs can access the data base in one of four modes. Serial access retrieves successive data entries from the data set. It is often used to process an entire data set in one pass. Direct access retrieves data entries based upon their record locations within the data base. It is generally used when the application already knows the identity of the data entry it wants. Calculated access retrieves the data entries in a master data set based on their search item (key) values. For example, calculated access might be used to obtain the data entry for an individual employee, given the employee number. Chained access is used to successively retrieve all the data entries in a detail data set that share a common search item value. For example, chained access would be used to retrieve all the

timecard data entries for a given employee. Data entries can be retrieved in either a forward or backward direction.

IMAGE/300 data bases are protected against unauthorized access by several layers of security features. In addition to the file system's domain passwords, the data base can be protected with a maintenance word that must be supplied to gain access to the data base from outside its domain. Data bases are stored as privileged HP 300 files, and cannot be accessed through the normal HP 300 file management system services.

Within a data base, individual data entries and data items are protected through an access level security scheme. Each data entry and data item can be optionally assigned a read access level and a write access level. When users or applications programs open the data base for access, they supply a level word, which determines the read and write levels they are permitted. For example, in the payroll data base of Fig. 3, certain users could be restricted to accessing only the PAY-PERIOD and TIMECARD data sets, while others might be permitted to access all data items except employee names and hourly rates, and still others might be permitted to access the entire data base.

Data Base Inquiry Facility

The data base inquiry (DBI) facility is used to access an IMAGE data base without an application program. It is particularly useful during application debugging to generate test data in the data base or examine data base contents. Data base inquiry is also useful for handling impromptu inquires into the data base, to display and summarize the data stored there. Using it, the user can display information about the data base structure, add data entries to a data set, delete data entries from a data set, modify data item values in a data entry, display the values of data items in selected data entries, and print data base inquiry responses on the printer.

Inquires to DBI take the form of natural, sentence-like commands. For example, to display the employee number and name of everyone in department 537 who is age 60 or older, the user would enter:

```
DISPLAY EMPLOYEE#, NAME FOR DEPARTMENT = 537 AND
                    AGE >=60
```

Data base inquiry would then display the employee names and numbers for the employees who met the selection criteria.

Data entries are selected for display by specifying values or value ranges for one or more data items. DBI accepts all the standard comparison relationships, and multiple selection criteria can be combined with AND and OR connectives to generate more complex inquiries. In response, DBI displays the contents of the qualifying data entries, or it can display only selected data items on request. All access through DBI is governed by IMAGE's standard security features, to prevent unauthorized access to sensitive information.

DBI is also a useful tool for data base maintenance. For example, if all the employees in job classification J4 have just received a pay increase to $3.60 per hour, the user can make the change with a single DBI command:

```
REPLACE RATE WITH 3.60 FOR JOB_CLASS =J4
```

To delete from the data base all employees who terminated employment over one year ago, again a single DBI command does the job:

```
DELETE EMPLOYEE FOR TERMINATION_DATE < 771231
```

DBI can also be used to add new data entries to the data base. For example, to add a new employee, the user would enter:

```
ADD EMPLOYEE
```

DBI requests a value for each data item in the data entry, by name, on the screen. The data supplied in response is checked for validity before being stored in the data base.

Acknowledgments

Numerous people have been involved in the design, implementation, and refinement of the HP 300 file and data base management subsystems. Araceli Keiser, Grant Shaw, and Doug Zumbiel deserve credit for evolving the file system from a concept to a product. Credit for many of the file management commands as well as the diagnostic and utility subsystem file capabilities goes to Al Dalrymple. Myron Zeissler deserves credit for all of the printer and disc I/O drivers. Tom Spross contributed the initial design and implementation of what grew to be a very comprehensive file management test tool. Credit for a file management performance tool goes to Bob Spivack. Bob Brown deserves credit for his efforts in developing IMAGE/300. Bob had responsibility for the schema processor and inquiry facility. We would like to express our appreciation to these individuals for their diligence and dedication.

Phillip N. Taylor

Phil Taylor has been developing software with HP since 1972. He was project manager for HP 300 file and data base management, and is now section manager for HP 300 data management and communications software. Born in San Francisco, he received his BS degree in mathematics in 1968 from the University of California at Davis, and was involved in computer systems development for four years before joining HP. Phil is married, has three children, and lives in San Jose, California. His interests include gardening, woodworking, backpacking, tennis, and basketball.

Alan T. Paré

Alan Paré was project leader for IMAGE/300. He received his BA degree in mathematics in 1967 from California State University at San Jose and his MS degree in applied mathematics in 1971 from the University of Santa Clara. Before joining HP in 1972 he specialized in information storage and retrieval systems and mathematical programming. With HP, he's contributed to the development of the RPG compiler for HP 21MX Computers. Alan is a native Californian, born in Santa Monica. He's single, has two children, and lives in Los Gatos. Among his interests are music, bicycling, and collecting art and antiques for his home.

An Easy-to-Use Report Generation Language

by Tu-Ting Cheng and Wendy Peikes

RPG-II (Report Program Generator II) is a widely used high-level problem-solving language for business data processing. The language is designed to facilitate production of well-formatted printed reports. It also greatly simplifies the tasks of data retrieval, file maintenance, and file creation.

In other commonly used compiled languages, such as BASIC, FORTRAN and COBOL, the programmer supplies the step-by-step instructions corresponding to the desired program logic. RPG-II differs in that the programmer need only describe the format of the input data, output reports and calculation operations. The RPG-II compiler does the rest, including supplying the program logic.

Another major difference between RPG-II and most other languages lies in the format of its source lines. Instead of the relatively free format of BASIC or FORTRAN, RPG-II is a completely fixed-format language. That is, each field on an RPG-II source line must appear in precisely the correct columns, and each set of columns may contain only one particular field. The descriptions of the fields corresponding to each set of columns are part of the language specification of RPG-II, and the names of these fields are printed across the appropriate columns on the RPG-II coding forms. Therefore, writing an RPG-II program consists simply of taking these coding forms and filling in the blanks.

RPG-II has extensive file processing capabilities. The file organizations supported include sequential, random, and keyed. A file in an RPG-II program may be processed either sequentially or randomly, and its records may be of either fixed or variable length. A portion of any keyed file can be processed sequentially or between two key limits, and records can be added or deleted from a keyed file at any time within an RPG-II program. There is also a technique called matching records, which allows the processing of multiple files as a single file. These are only a few of the advanced file processing techniques built into the language.

Options in RPG output statements relieve the programmer of the task of writing routines to format the output data. The RPG-II compiler generates all of the routines necessary to print and format heading, detail, and total output lines. The programmer only has to supply the variable parts of the information to be printed.

Similarly, the vast choice of operators available in the calculation statements allows for much flexibility in the type of tasks one can program without much effort. Again, the compiler supplies all of the routines necessary to perform all of the language's calculation actions, such as manipulating quantities and varying the course of events according to the results obtained.

Fig. 1 is an example of an RPG-II program. This program is very simple; all it does is read some input and echo this input data to a line printer.

```
PAGE     1 RPGII300  Vs. A.01.01     HEWLETT-PACKARD
SUN, MAY  6, 1979, 11:33 AM

   10H
   20FINPUT   IP  F       80
   30FOUTPUT  O   F       80
   40IINPUT   AA  01
   50I                                          1   80 DATA
   60OOUTPUT  D         01
   70O                          DATA       80

    0 ERRORS,     0 WARNINGS DETECTED DURING COMPILATION
```

Fig. 1. *A simple RPG-II program that reads some input and echoes it to a line printer.*

RPG-II/300

RPG-II/300 is the implementation of the RPG-II programming language on the HP/300 Computer System. Much of its design effort was spent ensuring compatibility with the RPG languages on the HP 3000 and the IBM System/32 and System/34. This allows existing programs from these machines to be transported to the HP/300 without extensive modification. It also enables programmers already familiar with RPG-II to use RPG-II/300 with very little additional training.

As a matter of fact, there is no need to make any changes to an RPG-II program from another machine. The HP 300 RPG language monitor and compiler together detect every possibily incompatibility in a source program. Each such error is either corrected by the language monitor or reported to the user so that it can be fixed.

This error detection occurs as a two-step process. First, the language monitor screens out incompatible source records. The program is then compiled, with the compiler on the alert for more language incompatibilities. For each such error, the user receives a message explaining what is wrong and how it can be corrected.

The language monitor finds all incompatibilities in a source program in the following manner:

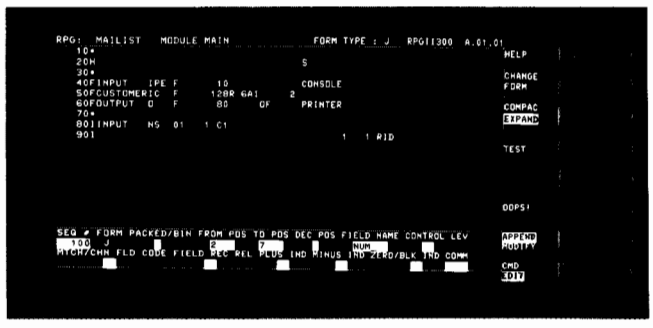


Fig. 2. *RPG-II/300 uses screen templates instead of coding sheets. Each field of the RPG specification is identified in the template area at the bottom of the screen. As each line is entered it takes its place in the display window above the template.*

Initially, an existing source program must be transferred to the HP 300 in the form of a sequential file. The user then brings the program into the language monitor via the COPY FILE command. This command performs line-by-line syntax checking on each source record in the program. If an error occurs, a self-explanatory error message immediately appears in the error message window, and the line in error is displayed. The user has two choices at this point: to ignore the error by hitting the BYPASS softkey, or to correct the error by typing in the changes. No matter what choice is made for each error, the user can be sure of one thing when the entire program has been brought in: that the program does not contain any unknown syntax errors.

Another feature of the RPG-II/300 compiler is that all error messages are self-explanatory, easily understood sentences. This type of human engineering, coupled with the features of the RPG-II/300 language monitor (such as the NEXT ERROR softkey), allow debugging convenience rarely found in other RPG systems.

The entire RPG-II/300 package, consisting of the compiler and the language monitor, adds up to a new approach for on-line development of RPG programs. The features of the language monitor are discussed in greater detail later in this article and in the article on page 9.

RPG Program Development

The RPG-II/300 language subsystem is similar to the Business BASIC/300 language subsystem in that program development can be performed in a highly interactive manner. The additional features of the source entry facility of RPG-II/300 greatly simplify the task of entering and editing a program. The most outstanding of these conveniences is the source entry window, which is in the form of a template. Each field in the RPG specification being entered is separately identified and labeled on the template. This eliminates the need for the programmer to count columns and spaces as the various fields are entered. Forward and backward tabbing and the IDS editing keys are additional features that make it easy to edit data in the templates.

As each source line is entered, it takes its place in its proper sequence in the display window (see Fig. 2). There is a different template for each of the distinct types of RPG-II specifications. To switch to another template, all the user need do is press the CHANGE FORM softkey and type in the new specification type, as shown in Fig. 3.

Each line in the program is checked for correct syntax as it is entered. Errors are reported immediately and can be corrected by typing the new field directly into the template.

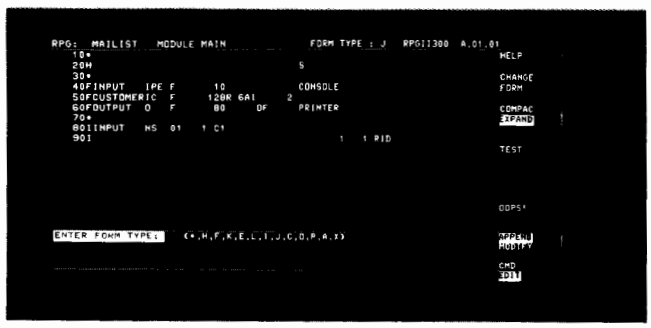


Fig. 3. *To switch templates the user presses the* CHANGE FORM *softkey, and is offered a menu of form types.*

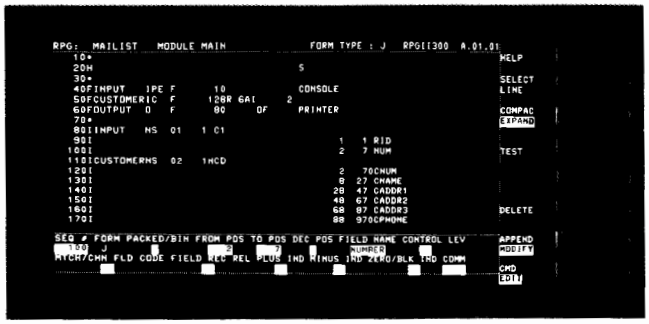


Fig. 4. *Editing a previously entered line using the* APPEND/MODIFY, SELECT LINE, *and* DELETE *softkeys.*

The user can also delete, edit, or duplicate a previously entered line via the following sequence of actions. First, hit the APPEND/MODIFY softkey to put the language monitor into modify mode. Then, enter the sequence number of the line to be edited, and hit the SELECT LINE softkey. The line selected will be displayed in the template window ready to be edited. Hitting the DELETE softkey causes the line to be deleted. If the line has been modified, hitting the ENTER key causes the replacement of the original line. If the sequence number has been changed, the line is duplicated in place on the screen (see Fig. 4).

To test the program, press the TEST softkey. The RPG language monitor automatically compiles, prepares, links, and executes the program, providing that no error has occurred. If the compiler has detected an error, the program listing is displayed in the upper window of the split screen. By pressing the NEXT ERROR softkey, the user can bring the line in error to the template window for editing as desired. The user can correct the error, or BYPASS it temporarily, or QUIT the NEXT ERROR processing. All this is done simply by hitting the appropriately labeled softkey (Fig. 5).

The RPG Language Monitor

The language monitor serves as the RPG-II/300 user interface both for source entry and for program compilation and testing. The integration of these functions, together with innovative features such as the source templates and split screen, make RPG-II program development on the HP/300 far more convenient than on any other machine.

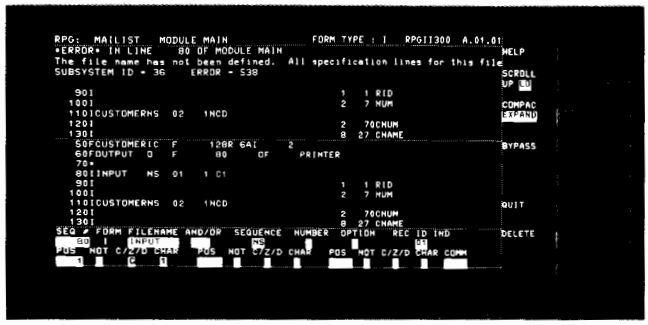


Fig. 5. *When the* TEST *softkey is pressed, compilation begins. If an error is discovered, it is displayed on a split screen. The user can correct the error,* BYPASS *it temporarily, or* QUIT *the error processing mode.*

The feature that does the most to facilitate program development is the source window, which is in the form of an RPG-II template. Since RPG is a language composed entirely of fixed-format statements, the user must enter each field on a source line in precisely the correct columns. This makes conventional RPG source entry a long, tedious, and error-prone process, with much time spent writing on coding sheets and counting columns. A program listing cannot even be read without a listing analyzer, which tells the programmer which columns correspond to which fields.

The language monitor's template facility takes care of all of this for the programmer. There is no need to count columns, or even use coding sheets. On the source entry template, each field on the source line has its own small window, marked clearly with the field's name. There is not even the need for the programmer to be concerned with right or left justification of fields, a frequent source of errors in a conventional RPG source entry system. The language monitor knows how each field should be justified, and does so accordingly. And, as soon as one field has been typed in, the language monitor positions the cursor to the next field to be entered on the source line.

The language monitor's immediate detection of the programmer's syntax errors saves much time and effort. In a conventional RPG source entry system, the whole program must be typed in and compiled before the programmer can be informed of any error. Not only does this waste much computer and programmer time by causing extra journeys through the source-entry/compilation/error-correction cycle, but it does nothing to prevent the programmer from making the same syntax error many times in a program.

The HP 300's language monitor, on the other hand, detects a syntax error as soon as the source line has been typed in. It informs the user of the error by placing a sentence in the message window that clearly explains precisely why the line is incorrect and what the programmer must do to fix the mistake. It also positions the source entry cursor to the field in error, so the programmer can easily type in the correction. However, the programmer is not forced to correct the error; hitting the BYPASS softkey will enter the source line, as is, into the program.

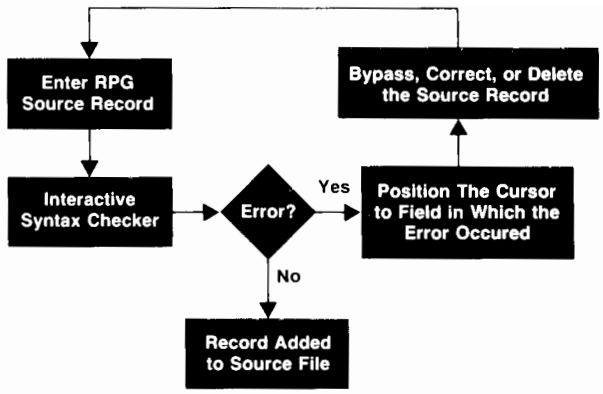


Fig. 7. *RPG-II/300 detects syntax errors immediately so they can be corrected interactively.*

This immediate reporting of syntax errors saves the program unnecessary trips through the compilation cycle, as well as preventing the programmer from making repeated mistakes. When the program has been compiled, the source entry screen splits to simultaneously display both the source program being edited and the list file created by the compiler.

Figs. 6 through 8 illustrate the operation of the language monitor and compare it with a conventional RPG system.

Multiple Terminals

RPG-II/300 features an interactive multiterminal data entry extension that significantly expands the terminal accessing capability of RPG. Using this extension, an executing RPG program can accept data entered from one or more HP/300 application terminals or from the IDS. The system automatically enacts this facility by starting a dialogue with the user at the IDS before the program executes. At this time, the user tells the system which terminal or group of terminals is going to be accessed by the program. This capability can also be used dynamically to add a terminal to or remove

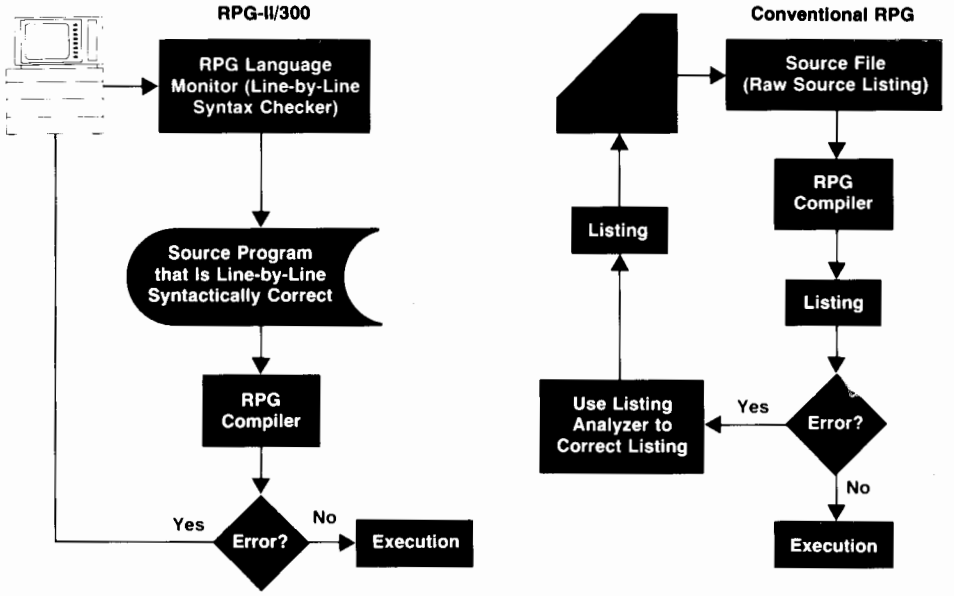


Fig. 6. *RPG-II/300 and conventional RPG program development cycle.*

Comparisons Of RPG Source Entry Approaches

RPG-II/300 Language Monitor	Conventional
1. No coding sheet is needed. The RPG templates provide descriptions of all fields, eliminating the need for column counting. The language monitor performs all necessary right and left justification and automatically sets a tab at the beginning of each field. This tool facilates source entry and minimizes syntax errors.	1. A coding sheet is required. The user must count the columns and keep track of the starting position of each field while entering the source.
2. The language monitor detects and reports syntax errors as soon as the source line is entered and positions the cursor to the column where the error occurred. The early error detection saves compilation time and the cursor positioning greatly simplifies the job of error correction.	2. The user must wait until the program is compiled to be informed of any errors.
3. This data entry facility minimizes the number of times that the user's program must go through the source entry loop.	3. Each time a single error is detected, the user's program must go through the entire source entry loop. Each time the code is corrected, there is a chance of introducing more errors.
	RPG-II/300 Features with No Conventional Counterparts
	1. Help facility
	2. Editor commands
	3. "Next Error" language monitor softkey
	4. Split-window screen to view source and complete listing simultaneously.

Fig. 8. *How the RPG-II/300 source entry method benefits the user.*

a terminal from an executing RPG program.

Such a powerful extension adds only minor changes to the RPG-II language constructs. All terminal operation is transparent to an RPG program, which views the terminal(s) as a conventional input file with the special device name CONSOLE. Terminal formatting and prompting for data are handled automatically on a field-by-field basis, according to the input specifications of the RPG program. No special coding effort is necessary; only the use of the device name CONSOLE is required to take advantage of this facility.

Wendy Peikes

Wendy Peikes received a BS degree in computer science and electrical engineering in 1976 from Massachusetts Institute of Technology, and an MS degree in computer science and computer engineering in 1978 from Stanford University. With HP since 1976, she's developed a syntax-directed editor for block-structured languages and contributed to the development of the RPG-II/300 compiler. Wendy was born in New York City and grew up there. Now living in Sunnyvale, California, she's single and enjoys racquetball, trap shooting, ice skating, and raising houseplants.

Tu-Ting Cheng

Born in Bangkok, Tu-Ting Cheng earned his BS degree in electrical engineering from National Taiwan University in 1969. Seven years later he received his PhD degree in computer science from Ohio State University. In between, he collected two MS degrees in different areas of computer science, one from the University of Wisconsin and the other from Ohio State. With HP since 1976, he's been involved with RPG-II/300 development, most recently as project leader. Tu-Ting is married and lives in Sunnyvale, California. His leisure activities include fishing and carpentry.

HP 300 Business BASIC

by May Y. Kovalick

BASIC IS THE ACRONYM for Beginner's All-Purpose Symbolic Instruction Code. As the name suggests, it is distinguished from other programming languages in its concern for the novice user. While BASIC is a general-purpose programming language, it is designed primarily to be easy to learn, easy to use, and easy to remember. Because of this, BASIC has found wide acceptance for educational, scientific, and commercial data processing. BASIC's simple statement format permits rapid development of simple, straightforward programs. In addition, its flexible input/output capability makes it well suited to interactive, terminal-oriented applications. BASIC was thus a natural choice for the HP 300.

Since an interactive environment facilitates learning, BASIC is oriented, but not restricted, to interactive use. BASIC/300 takes full advantage of the editing capabilities provided by the integrated display system (IDS) and the language monitor of the HP 300 to give the user an interactive program development environment. It also provides on-line syntax checking as each statement is entered, interactive error reporting, and symbolic debugging. All of these features provide many of the program development advantages of an interpreter.

On the other hand, because BASIC/300 is implemented as a compiler, it generates and stores developed programs in machine-executable form. This insures maximum runtime efficiency.

BASIC/300 is compatible with and is a superset of the ANSI X3.60 standard for minimal BASIC. Advanced features of the language, together with many enhancements in BASIC/300, allow programmers to accomplish more sophisticated tasks and make BASIC/300 a versatile business ap-

plication language.

BASIC/300 Data Types

The purpose of a program is to produce meaningful results by manipulating data, either numeric or character strings. BASIC/300 allows the user to represent data by meaningful alphanumeric names. A BASIC/300 numeric variable name is composed of an upper-case letter followed by any number of digits, lower-case letters, or the underline symbol _ (up to 15 characters are recognized). A string variable name is formed by attaching $ to the end of the valid numeric variable name. This feature makes programs more descriptive and understandable. BASIC/300 also supports five numeric data types: integer (16 bits), double integer (32 bits), short (32 bits floating point), real (64 bits floating point), and decimal. The inclusion of the decimal arithmetic data type enhances the usefulness of BASIC/300 for the manufacturing, inventory control, and commercial market. Instead of converting numbers to binary representation to perform calculations, arithmetic is done in base 10 by special routines. This allows the user to have more control over round-off effects and the number of significant digits.

The user is also given the ability to set the precision of individual BASIC decimal variables. Through a declaration, the user may specify both the total number of digits and the number of digits to the right of the decimal point. For example,

```
10 DECIMAL A[10, 2], B[9, 5]
```

declares that A has ten digits with two to the right of the decimal point, and B has nine digits, five to the right of the decimal point. A maximum of 27 digits is allowed.

Because of all the different data types available, mixed-mode arithmetic is provided by BASIC/300. Automatic data type conversion is done if necessary on arithmetic operations.

BASIC/300 also handles character string data composed of a sequence of valid ASCII characters. The string may be from 0 to 255 characters long. One may specify the maximum length of a string by using the DIM statement. For example,

```
10 DIM A$[25], B$(10) [7]
```

specifies that the string A$ may have up to 25 characters, and that each of the 11 elements of string array B$ may have up to seven characters. The default maximum length is 18 characters.

Strings may be concatenated with the concatenation operator &.

Normally a reference to a string refers to the entire string value. However, sometimes it is necessary to reference only a portion of the string. This kind of substring operation is allowed by using any of three different substring designators:

A$[m,n] specifies the mth through nth characters
A$[m;n] specifies n characters starting at the mth position
A$[m] specifies the mth through the last character.

These constructs make substring replacement and extraction possible.

There is also a set of over 30 built-in (or predefined) numeric and string functions available for reference by the BASIC/300 user. For example, POS(X$,Y$) provides the capability to do substring searching. It returns the starting position of the string Y$ within the string X$. Numeric/string conversion is possible with the VAL$ and VAL functions. These functions perform conversion of numeric data to string data and vice versa.

Branching to Alphanumeric Labels

Most BASICs allow users to branch to a statement or subroutine within the program with GOTO or GOSUB statements, respectively. The destination of the branch is usually identified by a line number. This means that the programmer either needs to know the line number for all the forward branches, or must go back and patch them up later. BASIC/300 allows users to label any statement with an alphanumeric label that has the same format as a variable name. This label may then be used to refer to the statement instead of the line number. This not only makes the program more readable and easier to follow, but also lets the programmer develop the program logically without worrying about line numbers.

Array Manipulations

An array is a collection of related data grouped under one name. The various values of the array are arranged in an ordered relationship. BASIC/300 users may use both numeric (all data types except decimal) and string arrays. Arrays may have up to 32 dimensions, and may be declared explicitly in a DIM or type declaration statement. For example, the statements

```
10 DIM A(-10:10)
20 INTEGER B (1:20)
```

declare that real array A has 21 elements, A(−10), A(−9),. . ., A(0), A(1), . . ., A(10), and integer array B has 20 elements, B(1) to B(20). These two statements explicitly specify the lower and upper bounds of the arrays.

Another way of specifying the lower bound of a dimension of an array is by using the OPTION BASE statement. It declares that all lower bounds of dimensions that are not explicitly specified are either 0 or 1. For example, the statements

```
30 OPTION BASE 1
40 DIM C(10), D$(5,5)
```

specify that array C has 10 elements, and that D$ has five elements in the first dimension and five elements in the second dimension, for a total of 25 elements. If a program unit does not contain an OPTION BASE statement and the array declaration does not explicitly specify a lower bound, the assumed lower bound is zero.

Another powerful tool to manipulate an array is the executable statement REDIM, which dynamically changes the shape of the array. Although the number of dimensions may not be changed, the number of elements in each dimension may be changed and new lower and upper bounds may be specified. Assuming that we are in the same program unit as above, the statements

```
100 REDIM D$(3,4)
110 REDIM C(-5:3)
```

redimension the arrays C and D$ so that D$ now has 12 elements, three in the first dimension and four in the second, and C now has nine elements.

Each element of the array may be accessed by using sub-

scripts, such as A(5)=B(7,2). The whole array may also be manipulated by MAT statements. These statements allow the user to initialize all the elements of the array, to input or print a whole array, and to do mathematical operations on numeric arrays.

Both numeric and string arrays are allocated space in the user's data space. The only limitation on the size of any numeric array is that the maximum number of elements specified in the DIM or type declaration statement may not exceed 32,767 divided by the data size of the array element. For string arrays, the maximum number of characters as specified in the DIM statement cannot exceed 65,536.

Functions and Subprograms

Increased interest in structured design and programming has made it desirable to be able to write modular programs. To accomplish this, BASIC/300 provides subprograms and multiline functions. Each function or subprogram is a block of statements that is complete in itself. In this way, programs may be developed in small modular units, and a more structured approach to BASIC programming is achieved. Multiline functions, subprograms, and the main program are known as program units. Each may be compiled independently, so that minor changes in a program unit do not require a complete recompilation. Parameters, either numeric or string, simple or array, or file numbers, may be passed to the subprograms or multiline functions.

Each subprogram is delimited by a SUB statement and a SUBEND statement. The number and types of parameters are defined in the SUB statement. Multiline functions are subprograms that return values. They are delimited by DEF and FNEND statements. The DEF statement is similar to a SUB statement in that it defines the number of parameters and their types. It also defines the type of the function, that is, the type of the value to be returned. Multiline functions are invoked by using the function's name and parameter list in an expression. Subprograms are invoked with a CALL statement.

Both subprograms and multiline functions may be recursive, that is, they may invoke themselves and they may invoke other subprograms or functions.

So far, we have mentioned one way to communicate between program units, namely, by passing parameters. Sometimes it is desirable to set up an area of data that is common to several program units. This is accomplished in BASIC/300 with the COM statement. Variables specified in a COM statement are placed in a common global area so that values assigned to these variables in one program unit are retained when control is transferred to another program unit. Thus data may be shared among program units without using a long list of variables passed as parameters.

Calling System Services

The AMIGO/300 operating system provides a wide range of system services for program management, task management, file management, data base management, synchronization, resource management, IDS control, and other functions. All of these services are useful and necessary for commercial business application programs.

Instead of inventing and designing a new language construct for every one of these services, BASIC/300 has provided a clean and easy statement that enables the user to take advantage of all the callable system services. This is the ICALL (intrinsic call) statement. The BASIC syntax for ICALL is

ICALL procedure-name (parameter-list)

or

ICALL procedure-name (parameter-list), numeric-variable

The procedure-name is the name of the system service procedure being called. The numeric-variable is used for calling procedures that return values. This statement allows the user to programmatically access all the facilities provided by AMIGO/300 without having to learn a completely new set of system-dependent language constructs for each of the system services needed.

The ICALL statement is versatile and easy to use, and helps to keep the BASIC language from being a conglomeration of complicated system-dependent commands.

File Manipulation

For applications that require permanent data storage external to a particular program, BASIC/300 provides a data file capability that allows flexible, direct manipulation of large volumes of data stored on files. There are many ways of arranging data in a file, depending upon the application. BASIC/300 supports four different file structures (sequential, direct, keyed-sequential, and memory), three types of file storage mechanisms in the files (ASCII, binary, and BASIC-formatted), and two methods of accessing the file (serial and direct). These provisions, plus programmatic file creation and purging, are directly available through the language constructs of BASIC/300.

Formatting Output

The PRINT USING and IMAGE statements of BASIC/300 give the user explicit and exact control over the format of program output. All types of numbers can be printed, and the exact positions of signs can be specified. String values can be printed in specified fields, and literal strings and blanks can be inserted whenever needed. Carriage return and line feed can also be under explicit control. Also provided are print functions such as TAB, SPA, LIN, and so on.

The PRINT USING statement allows easy construction of printed reports and formatted terminal displays. It may also be used to print to files or for formatting the printing of whole arrays. The format specification may be fixed at compile time with an IMAGE statement or a string literal, or may be fixed at runtime by PRINT USING A$; print-list. This allows the user to specify the format dynamically.

Take for example, the simple PRINT statement:

PRINT print-list

As mentioned above, we may specify the format with

PRINT USING format; print-list

If we want to print whole arrays, we may use

MAT PRINT [USING format;] print-list

The part in brackets is optional. To print to a file, we simply add the file number to the PRINT statement:

[MAT] PRINT #n [USING format]; print-list

This will do serial printing to the specified file starting at its current file pointer. We may also access the file directly by specifying the key of the record we wish to print to:

[MAT] PRINT #n, record-key [USING format]; print-list

These examples illustrate how a variety of output and file

manipulation tasks can be accomplished by making simple, easy extensions to the PRINT statement.

Compatibility

The core of BASIC/300 is compatible with the implementations of BASIC on the HP 9845A Desktop Computer[1] and the HP 250 Small-Business Computer.[2] There has been an earnest effort on the part of all of the developers of newer BASIC on HP machines to follow this trend, so that programs written on one HP machine are easily transportable to another.

Acknowledgments

Many people contributed to the success of BASIC/300. I would like to extend special thanks and appreciation to Denise Pitsch and Jon Kelley for their contributions to the design and implementation of the BASIC/300 compiler.

References

1. W.E. Eads and J.M. Walden, "A Highly Integrated Desktop Computer System," Hewlett-Packard Journal, April 1978.
2. D.L. Peery, "HP 250 BASIC: A Friendly, Interactive, Powerful System Language," Hewlett-Packard Journal, April 1979.

May Y. Kovalick

May Kovalick, project leader for the BASIC/300 compiler, holds BS degrees in mathematics and computer science from the University of California at Berkeley. She graduated in 1974 and has been involved in compiler development with HP since 1975. A native of Hong Kong, May is married and lives in Santa Clara, California. She devotes much of her time to Bible study and church activities. She also plays piano and enjoys reading and listening to music.

Innovative Package Design Enhances HP 300 Effectiveness

by David A. Horine

THE COMPACT HP 300 PACKAGE is a deceptively simple structure but it had to satisfy an extremely complex set of design requirements. One way to understand this is to examine the anatomy of the package on a layer-by-layer basis. The outermost layer, the skin of the machine, is the human interface. It has to be comfortable, relatively quiet, and relatively easy to use. It has to fit in an office environment and be durable for the conditions of use there, and it has to protect the user from any internal hazards. The next layer is the overall framework of the machine. This framework is governed by many requirements, such as manufacturability, interference protection, cost, and structural integrity. Next, there are the internal modules, such as the disc and power supply. Here, serviceability, component size, air cooling circuits, and grounding for electrostatic discharge are important parameters.

Another aspect of the machine is the electrical interconnection system, which includes the power cord, I/O cables, printed circuit boards, and the wire bonds to the integrated circuits. Cost, safety from shock hazard, international safety standards, durability, and immunity from electromagnetic interference (EMI) and electrostatic discharge (ESD) are important concerns. The printed circuit boards are also a level of packaging, with parameters of size governed by interconnect cost, packaging density, system partitioning, mechanical durability, tolerances to avoid misalignment, manufacturability, and serviceability. The innermost level of the machine is the circuit devices. Here, packaging design is concerned with orientation for optimum cooling, mounting, interconnection, and testability.

Altogether, the design parameters form a diverse and often conflicting set (Fig. 1). The HP 300 package design treated many of these parameters in new and innovative ways.

Human Factors

User-oriented design was an important consideration from the beginning. Fortunately, we were able to work closely with potential users in the form of the many software designers on the project team. This close interaction involved surveys, observation, and the construction of functional models that the software designers could use for program development. This work, together with literature and field surveys, led to the following features.

The overall configuration of the machine is basically a compact vertical arrangement of components. With this arrangement, the HP 300 occupies a minimal amount of floor space, which is often in short supply in the office environment (Fig. 2). The HP 300 can fit next to an existing desk or other work surface or an optional side table can be added to provide a wrap-around work area.

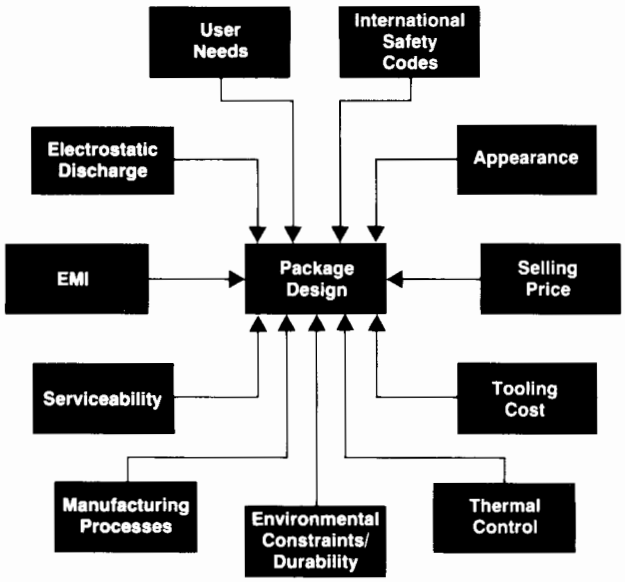


Fig. 1. *Design parameters for the HP 300 package design.*

The selection and arrangement of controls and displays were subjects for considerable research and deliberation. The most prominent of these features is the column of eight softkeys along the right side of the display. These keys are labeled under program control on the screen, and can be an extremely versatile tool in applications programs. While the concept of softkeys was axiomatic from the beginning of the project, the location was not. The vertical right-hand arrangement was selected because it offers maximum flexibility in creating labels. As many as three full lines of copy on the screen can be assigned to each key. The disadvantages to this arrangement were thought to be that it would require an excessive reach by the operator and that the right-hand labels could interfere with the length of other text on the screen. The first objection was overcome by providing an alternate method of accessing the softkeys by typing CONTROL 1 through CONTROL 8 on the main keyboard. Thus, knowledgeable user/typists can keep their hands on the keyboard while less familiar users can take advantage of the accuracy resulting from the close proximity of labels and screen-mounted keys. The second objection was met by providing the feature of horizontal scrolling. Thus, in the few cases where text and softkey labels overlap, it is possible to move the entire text to the left to uncover the hidden data.

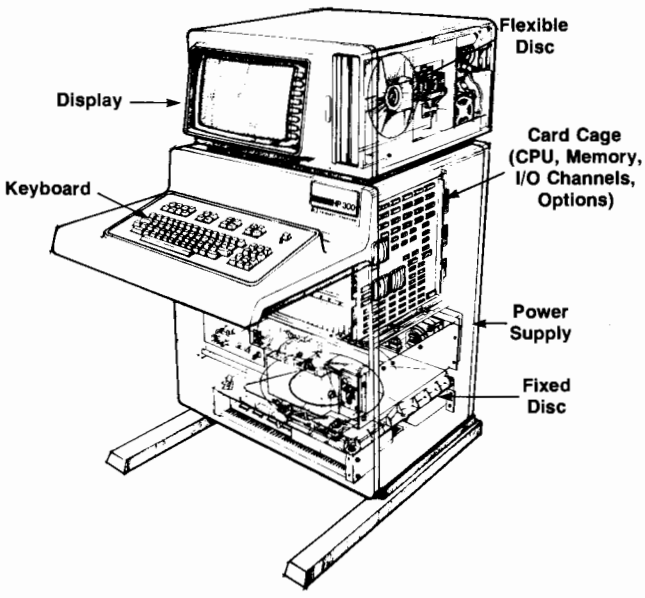


Fig. 2. *Compact vertical arrangement of components conserves space in the office environment.*

On the main alphanumeric keyboard, the arrangement of characters resembles that found on a typewriter. The philosophy here was that more people are familiar with a typewriter than with other alphanumeric keyboards, such as the teletypewriter. Keys have been added to this general arrangement to implement the full ASCII character set. Close cooperation among many HP divisions has resulted in this basic arrangement becoming a standard for a number of diverse data entry products.

Simplicity was a major consideration, and it resulted in the paring away of many extraneous controls and displays. Other than the CRT itself, the only display normally visible from the front of the machine is the ATTENTION light and thus the light's importance is emphasized. Several seldom-used controls and displays have been located behind a door to deemphasize their importance to the average user. The main power switch is located below the keyboard to deemphasize its relative importance in the on-going operation of the machine while providing easy accessibility in an emergency. Its location has the additional benefit that the switch is mounted directly to the power supply module so it can be removed and serviced with the supply.

We considered it important for the customer to be able to reconfigure and add boards to the card cage. This resulted in the development of a number of information labels on individual boards and on the card cage. These labels identify by names and colors the functions and locations of individual boards. In addition, the labels identify address select switches, cable locations, and diagnostic lights (Fig. 3).

The method by which the machine would be moved about raised some mechanical and human factors questions. A high degree of mobility was not thought to be necessary. On the other hand, the machine will be moved for servicing, floor cleaning, or other purposes. The obvious solution of four locking wheels was considered undesirable, since it would cause the anti-tip feet to be too high above the floor, creating a barrier for the operator's legs. Our solution was to mount two wheels in the rear of the machine and to use leveling pads in front. With this arrangement, it is easy to lift the machine by tilting the keyboard and then to move it as one would move a wheelbarrow. When the machine is set down, there is no possibility of accidental movement (Fig. 4).

Structural Design

Two of the most important parameters in the structural design of the cabinet were low cost and ease of modification. The second parameter arose from the need to accommodate the disc mountings and RFI/ESD requirements, which were not defined until late in the development pro-

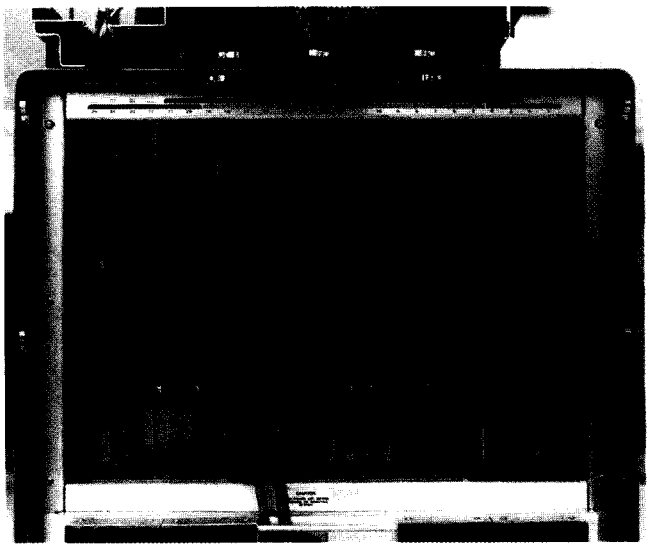


Fig. 3. *HP 300 card cage is labeled to identify the functions and locations of printed-circuit boards.*

cess. Our studies in cost reduction led us to examine the structures of home appliances, such as washing machines and refrigerators. Many of these products make use of the efficient monocoque design, which uses the skin of the machine as a part of the load-bearing structure. In other words, the skin serves a dual function of cover and structure, unlike other designs, in which it is common to use a skeletal structure covered by non-load-bearing shrouds. We

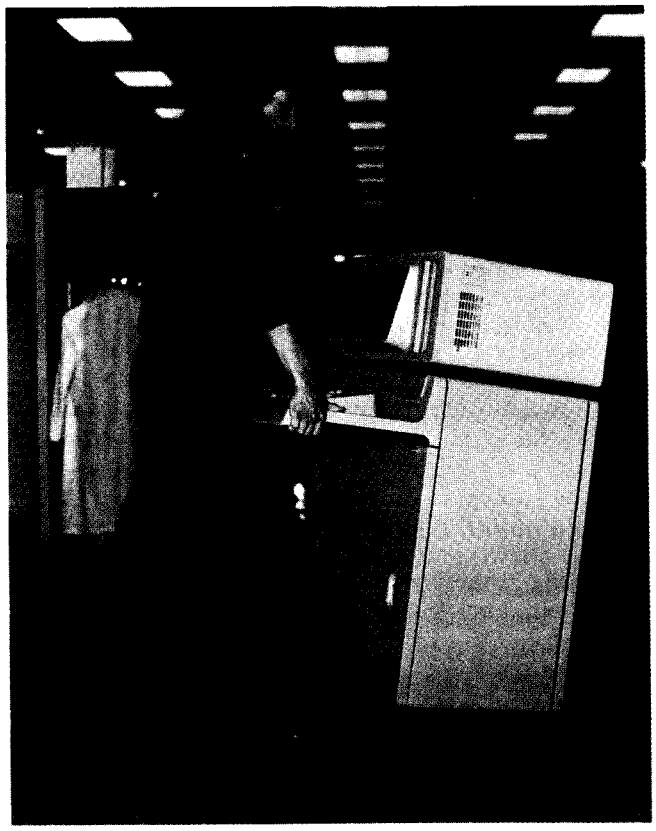


Fig. 4. *HP 300 can be moved like a wheelbarrow.*

A Novel Shipping Container

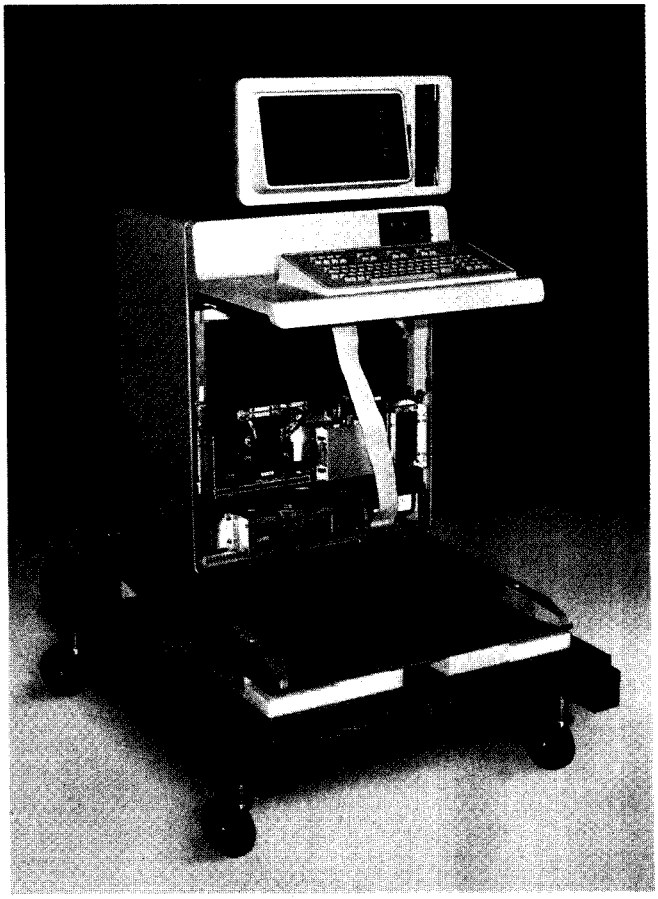
We are proud of the fact that the HP 300 shipping container system received the Best of Show award at the 1978 International Packaging Week Exposition. It is a multifunctional design that satisfies needs in production, shipping, and installation.

The packaging process starts with the insertion of caster wheels into the bottom of the shipping pallet. This pallet then serves as a mobile assembly platform and eliminates the need for conventional product handling systems. The empty minirack with feet attached is set on the pallet and the remaining parts are installed as it is moved from station to station in the production area. The wheel height was selected to provide comfortable access during assembly and testing.

The pallet also functions as an efficient shock and vibration isolator during shipping. Its floater base design incorporates two types of foam pads that serve as springs and dampers. Together, they reduce vibration loads and eliminate the problem of "bottoming out." These pads are also designed to withstand the extremes of hot and cold temperature cycling that each HP 300 undergoes in our production environmental test chamber.

After assembly and testing, the package is prepared for shipping by bolting the HP 300 to the pallet, adding accessory parts, installing a corrugated cover with snap-on/snap-off clips, and removing the wheels.

At the customer's site, the package is designed for quick disassembly and unloading. One inexperienced person can unpack and unload the 260-pound HP 300 in about five minutes. The unpacking instructions show a step-by-step procedure with pictures alone, so instructions in different languages are not required. After removal of the clipped-on cover and unbolting of the feet, a self-contained ramp is set alongside the pallet and the HP 300 is rolled off.



implemented this concept with a welded sheet steel frame that we call the "minirack" (Fig. 5). The minirack serves as the original part to which everything else is attached. The feet are bolted to the underside, the CRT/floppy disc assembly is riveted to the top, the keyboard shelf rivets to the front, internal modules including the card cage, disc, and power supply are bolted in, and a front and rear door are installed. Most of these modules mount in such a way that they enhance the rigidity of the overall structure. The process of adding parts to the minirack occurs while the minirack is in place on its shipping pallet with wheels mounted to the pallet's underside for production mobility (see "A Novel Shipping Container," page 28).

Most of the cabinet parts were designed so that they could be made on our numerically controlled punching machines. This "soft tooling" approach has the advantages of extremely low tooling cost and fast implementation of design changes. Consequently, while the basic cabinet design has remained from an early prototype stage, we were able to modify parts to include such things as cooling holes and grounding lugs without incurring the substantial time delays that often accompany the alternative of "hard" tooling.

Structural testing included a series of shake, shock, and drop tests that were defined as appropriate for the office environment. We also performed abnormal use tests, such as standing on the machine to change a light bulb. An unplanned measure of structural integrity occurred during a 1½-foot vertical drop test when the cabinet, after hitting the ground, continued to move and fell on its back side. The only significant damage was a twisted bracket and one loose connector. After reinstallation of the connector, the machine was operable.

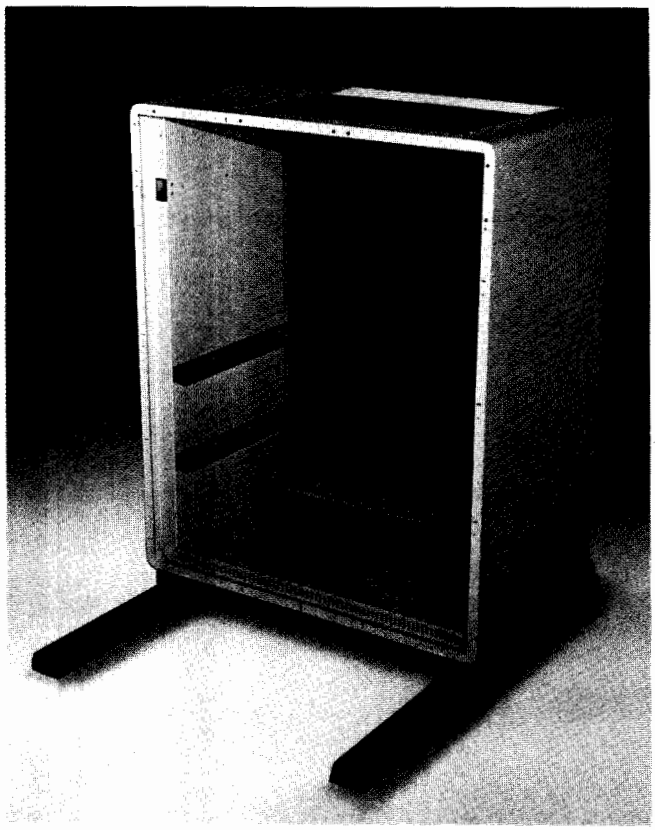


Fig. 5. *The HP 300's skin serves as both cover and structure. This "minirack" is the original part and everything else is attached to it.*

Serviceability

Rapid serviceability is an important consideration in terms of minimizing both user down time and service costs. The package design contributes to system serviceability by providing rapid access to internal components and by displaying a large amount of diagnostic and configuration information in the card cage. Access to all of the major modules can be gained by removing the front and rear doors, CRT shroud, and keyboard shroud. Most of these covers employ captive fasteners to minimize the common problem of lost screws. Once the rear door is unlocked and removed, the card cage is exposed. The cage contains all of the circuitry for the CPU, memory, and I/O devices, and most of the circuitry for the two discs and the integrated display system. Thus, most service functions can be performed in this one area.

As previously mentioned, the front side of the card cage boards contains many diagnostic lights and configuration switches. Thus it is often possible for the service person to make a quick assessment of system configuration and failure modes by looking at this area. Some subtle features that enhance serviceability include an auxiliary power connector on the backplane for connecting test probes, and a deliberate effort to minimize the number of fastener sizes.

Safety and Environmental Codes

Safety considerations played a large role in the HP 300 design, mainly because of an increasing awareness and definition of international safety codes. While many safety considerations can be identified by common sense, others can involve subtle misuse of the machine or the relative value judgments of safety agencies in different countries. Consequently, package design included ongoing reviews by our product safety group, and they spent much of their time keeping abreast of international developments. Our general philosophy was to design for the worst cases from all of the codes instead of tailoring machines for individual countries. Safety-related packaging features include:

- Definition of customer and service access areas. Service access areas require a tool for entry.
- Selection of materials to comply with various flammability standards.
- Caution and warning labels addressed to operators and service people.
- Shields for hazardous voltages and moving parts. Many of these shields are located inside the machine to protect the service person. Some of them also function to protect the system from shorting caused by such things as wandering screws and paper clips.
- Anti-tipover feet that also extend out from the rear of the machine to minimize the possibility that fan openings will be blocked by an adjacent wall.
- Numerous details to provide secure safety grounding throughout the chassis.

Two environmental concerns played a major role in package development: electromagnetic interference (EMI)

and electrostatic discharge (ESD). Limitations on EMI emissions are presently prescribed by the German communications authority. The HP 300 was tested to the German specifications early in its development and found to have excessive emissions. Application of some theory and much trial-and-error design led to the following changes:

- All conductive parts of the cabinet are grounded together at many points via screws and welds.
- Plated parts were substituted in places for painted ones to achieve better grounding.
- The doors are grounded to the main cabinet by leaf springs.
- I/O cable shields are grounded to the frame of the machine by die-cast mounting clips.
- Shields were added to several parts of the cabinet.

ESD problems occur when a person creates a spark by touching the machine or a nearby conductive object. In general, we found that ESD problems could be eliminated by solving EMI problems.

Acknowledgments

The contributions mentioned in this article represent the work of many people. Scott Stillinger did much of the product design in the form of configuration definition and parts design. Roger Lee was our industrial designer and the styling theme that he established with the HP 300 has been applied to a number of other compatible HP devices. Important production-oriented inputs were made by Virgil Springer and Tony Napolitan who joined the team first to assemble prototypes and later to function as supervisors in our manufacturing area. Curt Gowan contributed important ideas relating to serviceability. Bob Schaffer and Tony Napolitan established the basic concept of using the shipping pallet as a mobile assembly platform. Bill Kropf and Pat Wright were our shipping container designers. Beth Blomenkamp is to be commended for the prodigious task of producing over 500 square feet of mechanical drawings, which represented the package design documentation. Ron Morgan provided product safety consultation and review. This was an incredibly difficult task, since over fourteen different international safety standards were found to be applicable and many of these were being changed as the project progressed. Peter Rosenbladt contributed the foreign keyboard designs and much of the overall project coordination.

David A. Horine

Dave Horine was project manager for the HP 300 package design. Before coming to HP in 1971 he worked as a designer of surgical devices and later as a product design consultant. A graduate of Stanford University in mechanical engineering product design, he earned his BS in 1963, his MS in 1965 and the degree of engineer in 1969. For the past six years he has been an instructor in Stanford's Department of Design in addition to his work at HP. Originally from Glendale, California, Dave now lives in Los Altos with his wife and five daughters. Dave has an interest in long distance endurance sports, and his activities have included several 200-mile-per-day bike rides over Mt. Lassen, a ski trip across the Sierra Nevada range, running a marathon, and numerous backpacking trips. He also enjoys creative design activities such as jewelry and furniture building.

World-Wide Regulatory Compliance

The HP 300 Computer is marketed world-wide and must comply with a variety of domestic and international safety and other regulatory requirements. For North American markets, the product is listed and certified to the following safety requirements.

UL114: Standards for Safety of Office Appliances and Business Equipment (Underwriters Laboratories).

UL 478: Standard for Safety of Electronic Data Processing Units and Systems (Underwriters Laboratories).

CSA 22.2 No. 154: Data Processing Equipment (part of Canadian Electrical Code, Part II, Safety Standards for Electrical Equipment).

For the international market, the HP 300 is certified by the German agency Verband Deutsches Electrotechniker (Association of German Electrical Engineers) as conforming to the following standards.

VDE 0730: part I and part II-P: Regulation for Electric Motor-Operated Appliances for Domestic and Similar Purposes (Office Machines).

VDE 0871/6.78: Radio Interference Suppression of High Frequency Apparatus for Industrial, Scientific, and Medical (ISM) and Similar Purposes.

The HP 300 is licensed by the German FTZ (Bureau of Telecommunication Technology).

The HP 300 is designed for compliance with the following requirements of the International Electrotechnical Commission.

IEC 380: Safety of Electrically Energized Office Machines.

IEC 435: Safety of Data Processing Equipment.

The product is also designed for compliance with the safety standards of Finland, Switzerland, the United Kingdom, and Australia.

-Ronald E. Morgan

SPECIFICATIONS
HP 300 Computer System

HP 300 System Unit

The HP 300 System Unit is the central element in every HP 300 system configuration. It combines into a single, compact, integrated package all the hardware components necessary for system operation.

- Processor
- Main Memory
- Input/Output Channels
- 12M-byte Fixed Disc (optional)
- 1M-byte Flexible Disc Drive
- Integrated Display System
- Power Supply.

PROCESSOR:
- INSTRUCTION SET: 195 instructions
- DATA TYPES:
  - Bit
  - Byte
  - Integer (2- and 4-byte)
  - Floating point (4- and 8-byte)
- USER PROGRAM ADDRESSING SPACE:
  - Code: 2,064,384 bytes maximum (up to 63 code segments of up to 32,768 bytes each)
  - Data: 268,369,920 bytes maximum (up to 4096 data segments of up to 65,536 bytes each)
- MINOR CYCLE TIME: 270 nanoseconds; variable microcycle timing.
- LEVELS OF INTERRUPT PRIORITY: 15.

MEMORY
- WORD LENGTH: 22 bits (16 data/6 error correction)
- CYCLE TIME: 500 nanoseconds
- MEMORY MODULE: 128K bytes
- MINIMUM MEMORY: 256K bytes (2 modules)
- MAXIMUM MEMORY: 1024K bytes (8 modules)
- MEMORY TECHNOLOGY: 16K-bit MOS RAMs

GENERAL INPUT/OUTPUT CHANNEL (GIC)
- CAPACITY: 8 devices per GIC, maximum; 2 GICs per system, maximum
- DATA TRANSFER RATE: 1M-byte/second maximum
- CABLE LENGTH: 15m (50 ft) maximum per GIC
- INTERFACE: General-purpose byte-parallel interface bus
- DEVICES SUPPORTED:
  - Integrated Display System
  - Fixed Disc
  - Flexible Disc Drive
  - 2631A Serial Printer
  - 7906M/S Disc Drive
  - 7920M/S Disc Drive
  - 7925M/S Disc Drive

ASYNCHRONOUS DATA COMMUNICATION CHANNEL (ADCC)
- CAPACITY: 8 devices per ADCC, maximum (4-Main, 4-Extender); 2 ADCCs per system maximum (total of 16 devices per system)
- DATA RATES: 50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 2400, 4800, 9600 baud
- CABLE LENGTH: 15m (50 ft) maximum per device
- INTERFACE: RS-232C/CCITT V.24 asynchronous, bit-serial interface
- DEVICES SUPPORTED:
  - 2640B Interactive Display Terminal
  - 2645A Display Station
  - 2647A Intelligent Graphics Terminal
  - 2648A Graphics Terminal

FIXED DISC (HP 7910K-020)
- CAPACITY: 12 million 8-bit bytes (formatted)
- BYTES/SECTOR: 256
- SECTORS/TRACK: 32
- TRACKS: 738 × 2
- TRACKS/INCH: 300
- BITS/INCH: 3225
- TRACK-TRACK SEEK TIME: 10 ms
- AVERAGE SEEK: 70 ms
- WORST CASE SEEK: 110 ms
- AVERAGE LATENCY: 10 ms
- TRANSFER RATE: 526.5 kilobytes/second

FLEXIBLE DISC DRIVE (HP 7902)
- CAPACITY:
  - 1.03M bytes (1,029,120 bytes)
  - 256 bytes/sector
  - 30 sectors/track
  - 67 tracks/surface
  - 2 surfaces
- MEDIUM: 2-sided, double-density flexible disc (IBM #2736700 diskette)
- TRACK-TO-TRACK SEEK TIME: 18 ms
- AVERAGE SEEK TIME: 91 ms
- AVERAGE LATENCY: 83 ms
- DATA TRANSFER RATE: 100K bytes/s (burst)

INTEGRATED DISPLAY SYSTEM (IDS)
- DISPLAY SCREEN DIMENSIONS: 13.7 × 26.4 cm (5.4 × 10.4 in); 24 lines of 80 characters.
- CHARACTERS: 2.46 × 3.18 mm (.097 × .125 in); 7×9 enhanced dot matrix with half dot shifting.
- INTENSITY CONTROL: Operator accessible
- STANDARD CHARACTER SET: 128-character USASCII
- OPTIONAL CHARACTER SETS: Math characters, Line drawing set, Large characters, International characters.
- DISPLAY ENHANCEMENTS: Half-bright, Blinking, Inverse video (black-on-white), Underline.
- KEY-CONTROLLED FUNCTIONS: Display enhancements, Character set selection, Set/clear tab, Display of control functions, Screen hardcopy.
- WINDOWING
  - Concurrently Active Windows: 32 maximum
  - Borders: 1 arbitrary vertical border maximum; Arbitrary horizontal borders.
  - I/O: Input from one displayed window at a time; Output to any active window on an asynchronous basis.
  - Format Modes: Unformatted (window contents modifiable from keyboard); Implicitly Formatted (output protected from modification); Explicitly Formatted (input restricted to defined fields).
- FILE ATTACHMENT/SCROLLING/EDITING
  - Files Attachable: Keyed Sequential with DOUBLE keys, Direct.
  - Maximum File Length: Arbitrary
  - Maximum Record Length: 160 displayable characters (256 including control characters).
  - Scrolling Functions: Scroll up/down, Scroll left/right, Display first/last page, Display previous/next page.
  - Editing Functions: Scrolling, Cursor up/down/left/right, Character replace (type over), Character insert/delete, Line insert/delete.
- SOFTKEYS
  - Number of Softkeys: 8.
  - Softkey Labels: Dynamic labelling; 1 to 3 lines/label; 1 to 80 characters/label line.
  - Softkey Mode: Terminating (input terminates when key is struck); Non-terminating (input continues after key is struck).

ENVIRONMENTAL
- TEMPERATURE
  - Operating: 10° to 40° C (50° to 104° F)
  - Non-Operating: −40° to 65° C (−40° to 149° F)
  - Maximum Rate of Change: 10° C/hour (linear)
- HUMIDITY
  - Operating: 20% to 80% RH (maximum wet bulb temperature 26° C, no condensation)
  - Non-Operating: 8% to 80% RH (maximum wet bulb temperature 30° C, no condensation)
- ALTITUDE
  - Operating: To 4600m (15,000 ft)
  - Non-Operating: To 15,200m (50,000 ft)

GENERAL
- POWER REQUIREMENTS
  - Voltage: 100, 120, 220, or 240V; ±5%, −10%
  - Frequency: 50 or 60 Hz; +2 Hz, −2 Hz
- POWER DISTURBANCES (SYSTEM WITHSTANDS):
  - Short-Term Undervoltage: <85% of nominal for 11 ms duration, measured from peak of the ac waveform.
  - Short-Term Overvoltage: 150V for 30 s duration (110, 120V); 300V for 30 s duration (220, 240V)
  - Line Interference Pulses: 1000V for 50 μs (line-to-neutral or neutral-to-ground)
  - Fast Pulse Disturbances: 1500V for 30 ns (line and neutral to ground)
- SHOCK (NON-OPERATING): 30 g
- FLEXIBLE DISC STORAGE TEMPERATURE: 5° to 50° C (41° F to 122° F)
- MAXIMUM RATE OF TEMPERATURE CHANGE: 20°C/hr (36°F/hr)
- HUMIDITY: 8% to 80% RH
- SAFETY/RFI CERTIFICATION
  - USA: UL 478, 114
  - Canada: C22.2 #154

HP 300 Software

AMIGO/300 OPERATING SYSTEM

AMIGO/300 FILE SYSTEM
- FILE STRUCTURES: Sequential, Relative, Keyed Sequential, Direct (Hashed), Library, Primitive, Memory, Null.
- ACCESS METHODS: Serial (forward/backward), Keyed.
- KEY TYPES SUPPORTED:
  - Integer (2- and 4-byte)
  - Real (4- and 8-byte)

Character string (to 255 bytes)
KEYED RETRIEVAL MODES: Exact key match, Next key < search key, Next key > search key, Next key <= search key, Next key >= search key.
FILE SHARING MODES: Exclusive access, Read-only sharing, Update sharing with file locking/unlocking.
MAXIMUM RECORD LENGTH: 2028 to 2036 bytes, including keys, depending on file structure.

IMAGE/300 DATA BASE MANAGEMENT SYSTEM (OPTIONAL)
DATA SET STRUCTURE: Networked, with master and detail data sets.
DATA SET ACCESS METHODS: Serial, Direct, Calculated (Master set), Chained (Detail set).
DATA ITEM NAMES PER DATA BASE: 255 maximum
DATA ITEMS PER DATA ENTRY: 127 maximum
DATA SETS PER DATA BASE: 50 maximum
DETAIL DATA SETS PER MASTER DATA SET: 8 maximum
SEARCH ITEMS (KEYS) PER DETAIL SET: 8 maximum
DATA ENTRY SIZE: 2034 bytes (Master), 2020 bytes (Detail) maximum
DATA ENTRIES PER DATA SET: 65,535 maximum
DATA ENTRIES PER CHAIN: 65,535 maximum

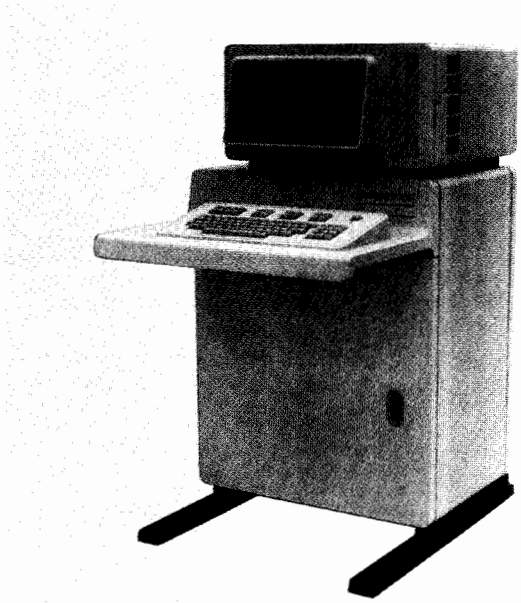
BUSINESS BASIC/300

RPG-II/300

UTILITIES
SORT/MERGE
Input Files: 16 maximum
Input File Organizations:
Sequential
Relative
Keyed Sequential
Direct
Sort/Merge Keys: 16 maximum
Sort/Merge Key Types:
ASCII string (to 255 bytes)
Logical (2 bytes)
Integer (2 or 4 bytes)
Real (4 or 8 bytes)
Packed Decimal (to 27 digits)
Zoned Decimal (to 27 digits)
Key Positions: arbitrary
Sorted/Merged Output Options:
Complete records
Sorted/merged keys only
Record addresses (tag sort)—Sort only
TYPIST
HELP
SYSTEM BUILD
SYSTEM STARTUP
DIAGNOSTIC AND UTILITY SYSTEM

PRICE IN U.S.A.: $36,500 (includes one language).

MANUFACTURING DIVISION: GENERAL SYSTEMS DIVISION
19447 Pruneridge Avenue
Cupertino, California 95014 U.S.A.



Hewlett-Packard Company, 1501 Page Mill Road, Palo Alto, California 94304

HEWLETT-PACKARD JOURNAL

Bulk Rate
U.S. Postage
Paid
Hewlett-Packard
Company

CHANGE OF ADDRESS: To change your address or delete your name from our mailing list please send us your old address label. Send changes to Hewlett-Packard Journal, 1501 Page Mill Road, Palo Alto, California 94304 U.S.A. Allow 60 days.