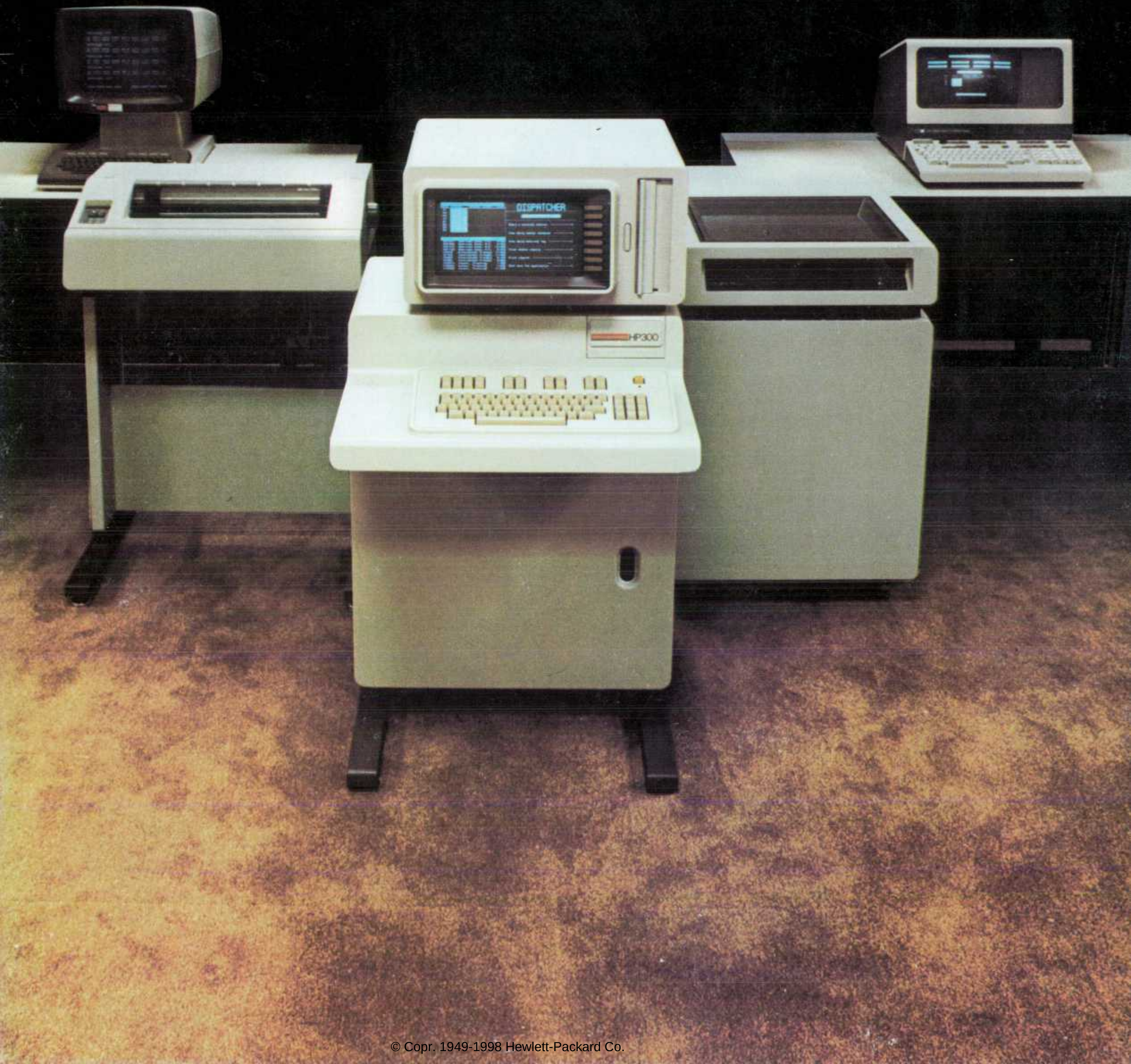
JULY 1979

HEWLETT-PACKARD JOURNAL



HEWLETT-PACKARD JOURNAL

Technical Information from the Laboratories of Hewlett-Packard Company

JULY 1979 Volume 30 • Number 7

Contents:

In this Issue:

This is the second of two successive issues devoted to the HP 300 Computer. This all-new state-of-the-art machine is the forerunner of a computer family that's expected to meet the needs of small business users into the 1980s.

Last month's articles introduced the HP 300 and described it from the user's point of view—its novel keyboard and display system, what it's like to program, its packaging and software. This month we go inside and look at its architecture and processor design (p. 3), its input/output system (p. 9), the hardware design of its display system (p. 13), its software operating system (p. 17), and its power supply (p. 25).

Central to the processor design story are three custom silicon-on-sapphire large-scale integrated circuit chips, each containing many thousands of transistors. Circuits can be packed so densely on these chips that it was possible to build the HP 300's central processing unit on only two printed circuit boards, compared to the eight boards that would have been required for equivalent capability using older technology. The power supply is of interest because it's an off-the-shelf HP product designed for general computer mainframe applications, and not, as is more often the case, a special unit that can't be used in any other computer.

This month's cover shows the HP 300 with some of the peripheral equipment it can support. One HP 300 system unit will support up to 16 applications terminals, two printers, and external disc memories with a capacity of 260 million characters.

-R. P. Dolan

Cost-Effective Hardware for a Compact Integrated Business Computer

CMOS/SOS technology helps reduce an eight-board processor to only two boards. Advanced architecture supports the features the user sees.

by Arndt B. Bergh and Kenyon C. Y. Mei

THE HP 300 SOFTWARE ENVIRONMENT, described last month, is built on the architectural capabilities of the HP 300 computer hardware. The HP 300 is a stack machine with additional data spaces and a separate code space. This means that the environment of an executing program consists of a non-modifiable code space of one or more code segments, and a number of data spaces that include the stack, the global data segment, and additional array data segments that may be needed by the program (see Fig. 1). The stack is used for program control, parameter passing, local storage, and expression evaluation. The data segments provide space for global data and additional array data.

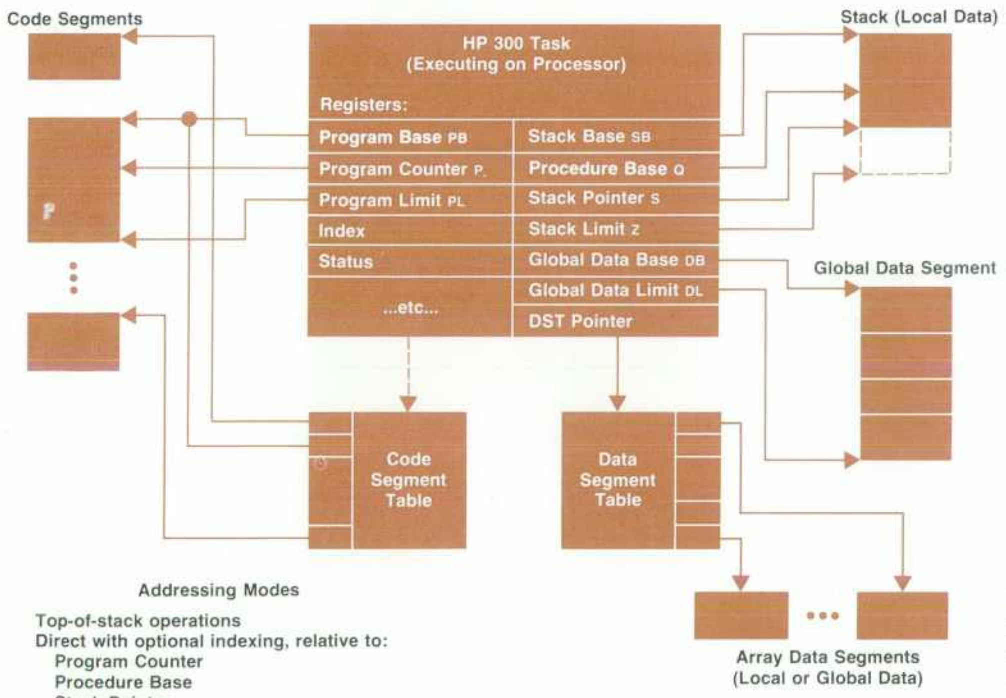
There is a stack for every program or task, but only one is active at any time. The stack operates in the normal push down, pop up mode using zero-address instructions. One-address, indexed, direct or indirect instructions are provided for access to data within the stack, global, and code segment spaces. Indirect reference addressing is provided for access to data in data segments.

The HP 300 has a rich instruction set of almost 200 instructions, including data manipulation instructions, program control instructions, and privileged instructions to aid the task of the operating system.

Capabilities provided by this structure include relocatability, reentrancy, recursion, code sharing, program protection, convenient dynamic storage allocation, and a logical virtual memory structure for code and data that provides a very large machine address space.

All program and data spaces are relocatable, that is, they can be located anywhere in real memory without alteration. A natural reentrancy capability is provided since program code is separated from data and is not alterable. This means that a program may be interrupted from a code segment, and that segment may be used by other programs and later reentered by the first without concern that the code may have been altered by the other programs.

By a simple linking mechanism, code is accessed logi-



Addressing Modes

Top-of-stack operations
Direct with optional indexing, relative to:
- Program Counter
- Procedure Base
- Stack Pointer
- Global Data Base

Indirect via above, with optional indexing to:
- Current Code Segment
- Any Data Segment

Fig. 1. *HP 300 addressing scheme. The environment of an executing program includes non-modifiable code segments and a number of data spaces that include the stack, the global data segment, and array data segments as needed.*

cally through system tables. Together with the previous two properties, this provides a convenient means for the sharing of code. These code segment properties and the procedure call mechanism that is used with the stack also provide a natural recursion capability, that is, the ability of a segment of code to call itself.

Both user and system code and data are subject to memory protection checks, a feature that is very important to the integrity of the system when used in multiprogramming and multitasking applications. Dynamic storage allocation is a feature of the stack architecture that provides a more efficient use of main memory.

Virtual Memory

An architectural feature that greatly expands the capability of the HP 300 is the new virtual memory structure that includes both code and data segmentation. Tables maintained by the system contain entries for all the code and data segments used by a program. Only the code and data segments currently being used need be present in memory, with new segments being brought into memory as they are needed. The operating system uses this feature to develop a working set of code and data segments for each program so the use of memory resources can be optimized.

Data Segmentation

A variable may be passed to a program or procedure either by value or by reference, that is, by passing its actual value or by passing a pointer to the storage location of its value. In the HP 300, the indirect pointer used for accessing reference variables has been made a 32-bit data descriptor, or label. This label specifies the data area in which the data resides, and the relative location within that data area. If the label points to a data segment, the segment number in the label is used to index into the data segment table to find the location of the segment in real memory (see Fig. 2). The offset in the data label plus the index register then are used to compute the relative address of the data within the segment. For protection, additional information in the table is used to verify the validity of the label. This label structure provides the HP 300 with addressability to a data space of up to 250 million bytes for the system plus up to 250 million additional bytes of data for each task that is active on the system.

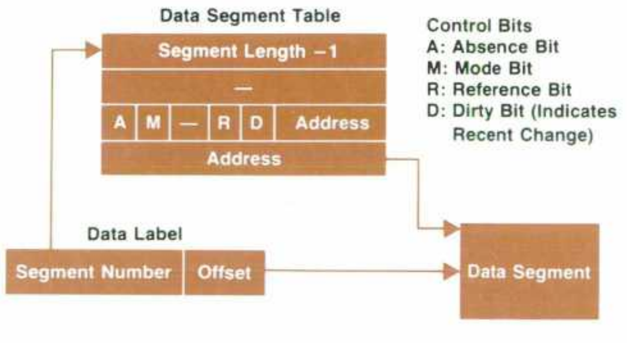


Fig. 2. *When variables are passed by reference to a memory location, the pointer to the location is a 32-bit data label that contains a segment number and an offset. The segment number is used to index into the data segment table for the status and location of the segment. Segment presence and mode are checked and the offset is added to the address from the table to give the address of the data.*

Code Segmentation

The HP 300 system programming language is a block-structured, procedure-oriented language. Procedures used by a program may be located in separate code segments. Calls to these segments are made through 32-bit program labels that consist of a segment number and a logical entry number in the target segment. The segment number is used to index into a system-maintained code segment table to find the location of the target code segment. The logical entry number in the label is used to index into the procedure entry point table (STT) appended to the code segment to find the starting address of the procedure. This segment structure provides up to a gigabyte of system code space plus up to a gigabyte of additional user code space for each task that is active on the system.

Instruction Set

Instructions are provided for manipulating the following data types: bit, byte, decimal and byte strings, 16-bit integer and logical, 32-bit integer and floating point, and 64-bit floating point. These are predominantly zero-address instructions, such as add, subtract, compare, shift, and so on. They operate on the top of the stack and their opcode-only nature saves space and thus improves code compactness. Indexable one-address memory instructions, such as quad load, are provided for accessing these data types in memory. Program control instructions, such as procedure call, are included to support the block-structured system programming language. A set of privileged instructions for I/O and operating system use, such as task launch, also have been provided.

All these features provide advantages that are important in a multiprogramming environment. The user program lives in a protected logical addressing structure, with program space managed and protected by the system. Although it is a 16-bit machine, the HP 300 provides processing power approaching that of a 32-bit machine.

Central Processing Unit

The HP 300 CPU, Fig. 3, is an answer to the challenges of high computing speed, low power consumption, and physical compactness. Three custom-designed large-scale integrated (LSI) circuits, processed by HP's complementary metal-oxide-semiconductor/silicon-on-sapphire (CMOS/SOS) facility, enabled our logic designers to pack what might have been an eight-board CPU into two. The integrated-circuit chip set, 6K 32-bit words of read-only memory control store, and some peripheral logic reside on one of the boards, the processor board. The other board, the bus interface controller board, contains registers, drivers, and the asynchronous hand-shake logic required to communicate through the intermodule bus, which connects the processor elements (Fig. 4).

The fact that the HP 300 CPU is a microprogrammed processor greatly simplifies hardware design and allows a great deal of flexibility in the development schedule. It also allows other groups to take advantage of this powerful processor. For example, the input/output processor is emulated by the CPU, that is, a portion of the CPU's microprogramming implements a separate I/O processor with an instruction set that is different from the CPU's. Another new HP computer system, the HP 3000/33, uses the same proces-

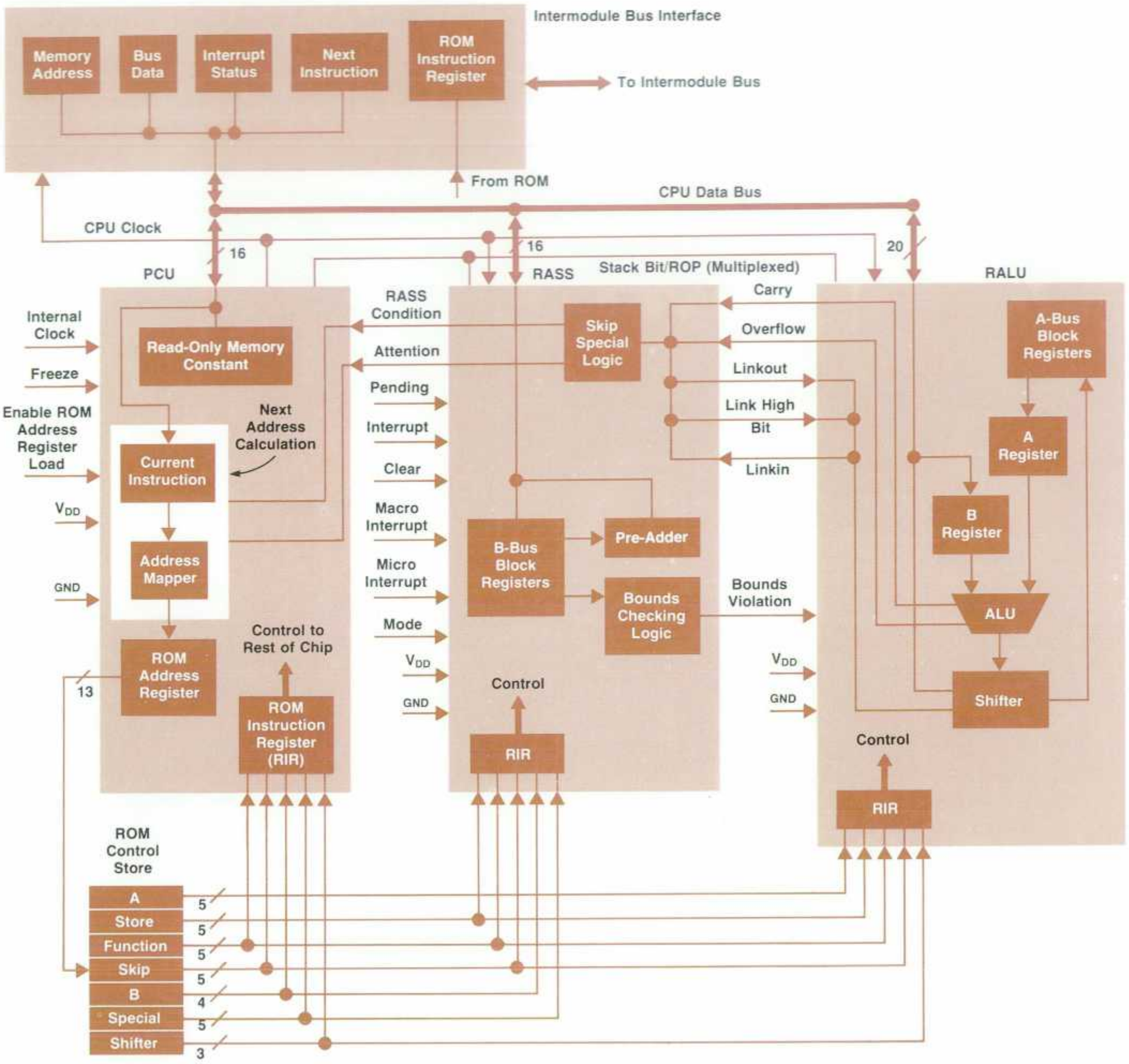
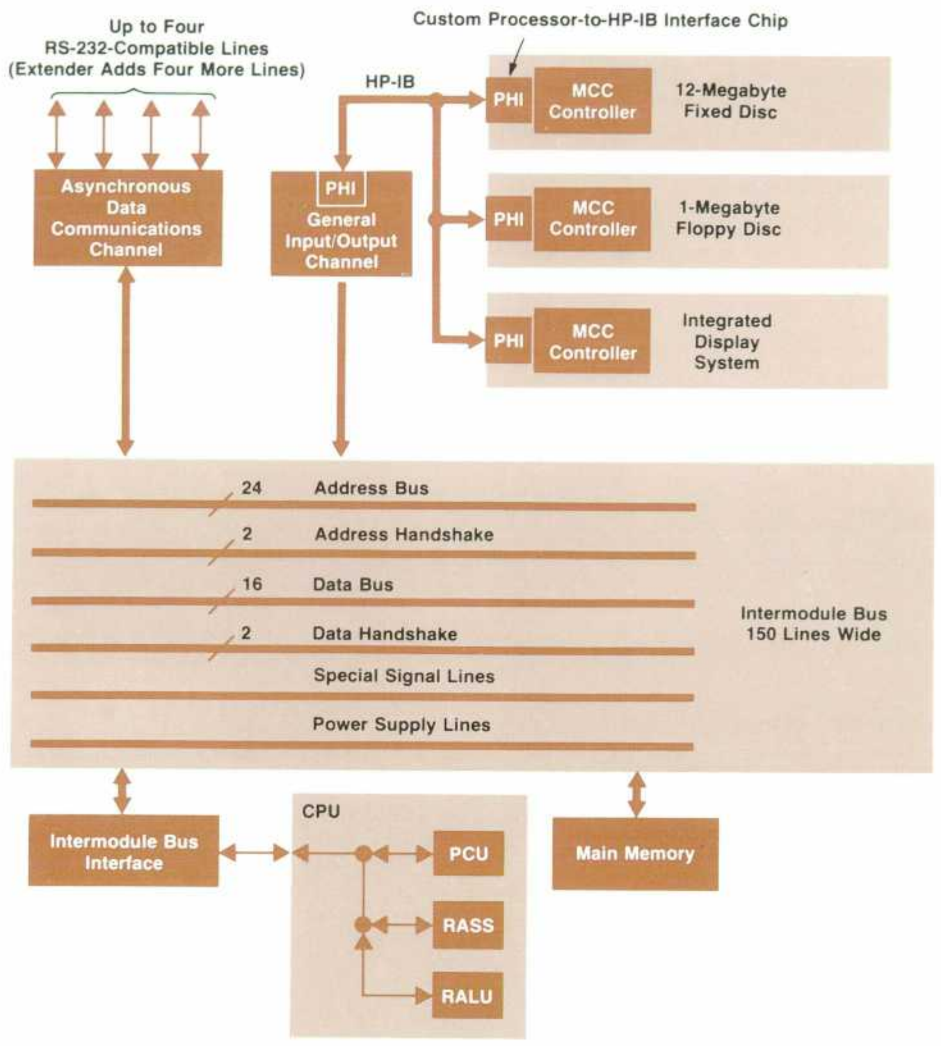


Fig. 3. *The HP 300 CPU hardware is functionally partitioned into three CMOS/SOS chips. The processor control unit (PCU) chip generates microinstruction addresses that control the other two chips: the register, address, skip, and special (RASS) chip and the register, arithmetic, and logic unit (RALU) chip.*

sor hardware (with one pin hardwired to a different voltage) with its own microprogram. A powerful self test is easily implemented by the inclusion of self-test microcode.

CMOS/SOS technology was chosen mainly because of its good speed-power product, high density, and ease of design. This technology allowed relatively inexperienced logic designers to start designing while the process was being developed. This is because designing a CMOS chip is like designing static logic, so the designers did not have to be concerned about charge storage and timing, which are critical in dynamic NMOS integrated circuits. The engineering decision to design a three-chip set was based on optimal logic partitioning, pin limitation, projected yield due to chip size, speed, and available development resources such as manpower and time.

The HP 300 CPU is a general-purpose microprogrammed processor with special features useful for emulation of the HP 300 system language instructions. Decoding of Huffman-coded instructions and automatic bounds checking are all done by hardware. Two top-of-stack (TOS) registers are provided for fast access to the stack. To reduce the complexity of the stack control logic, the stack area in main



Up to Four RS-232-Compatible Lines (Extender Adds Four More Lines)
Custom Processor-to-HP-IB Interface Chip
HP-IB
PHI
MCC Controller
12-Megabyte Fixed Disc
Asynchronous Data Communications Channel
PHI
General Input/Output Channel
PHI
MCC Controller
1-Megabyte Floppy Disc
PHI
MCC Controller
Integrated Display System
24
Address Bus
2
Address Handshake
16
Data Bus
2
Data Handshake
Special Signal Lines
Power Supply Lines
Intermodule Bus 150 Lines Wide
CPU
Intermodule Bus Interface
PCU
RASS
RALU
Main Memory

Fig. 4. *The 150-line intermodule bus connects the processor elements, the memory elements, and the input/output channels. The bus is asynchronous and provides separate address and data handshake lines for better performance. Peripheral devices are connected to the general input/output channel via the HP Interface Bus (IEEE 488).*

memory is always kept current, that is, the contents of the hardware stack registers are duplicated in main memory. All data registers are 16 bits wide. Double- and four-word instructions are facilitated by taking advantage of two- and four-word shift hardware. Environmental registers (address pointers) are 20 bits wide for easy manipulation of 20-bit addresses.

The LSI Chip Set

The processor control chip (PCU), Fig. 5, is the smallest chip (5260 × 3570 micrometres). It contains 5000 devices (transistors and diodes). Microprogram sequencing, CPU clock generation, and a real-time clock are the major functions of the PCU. The sequencing hardware includes microaddress incrementers, a two-level subroutine save-register stack, and a mapper that maps the current instruction to the beginning of the microprogram that executes it. The PCU generates a variable, single-phase microcycle clock that governs CPU operation. The length of the clock period depends on the microinstruction being executed. It can also be extended to wait for memory data or the next microinstruction during a jump. A system real-time clock and immediate microprogram data are also generated by the PCU.

The register-address-skip-special (RASS) chip, Fig. 6, and the register-arithmetic-logic-unit (RALU) chip, Fig. 7, together perform the data path functions. The RALU chip contains about 8000 devices and measures 4930 × 4930 micrometres. It provides 16 registers, eight for address storage and the other for general-purpose use. The arithmetic and logic computation hardware also reside on this chip. In addition to standard ALU functions, the RALU performs integer multiply/divide, decimal addition, and 64-bit shift operations.

The RASS chip (4480 × 5200 micrometres, 7000 devices) serves several purposes. Its register file provides the second operand to the ALU. If the current instruction is a memory reference type, special hardware will extract the Huffman-coded displacement with the contents of the index register added to it if necessary. The RASS also automatically checks the effective address against its proper base and limit registers within one microcycle. This powerful feature saves both time and microprogram steps. Condition code, interrupt priority control, and skip decision logic also reside on the RASS.

The Main Memory

A standard HP 300 Computer System includes 256K bytes of main storage, which is expandable to 1024K bytes in 128K-byte increments. The main memory consists of a

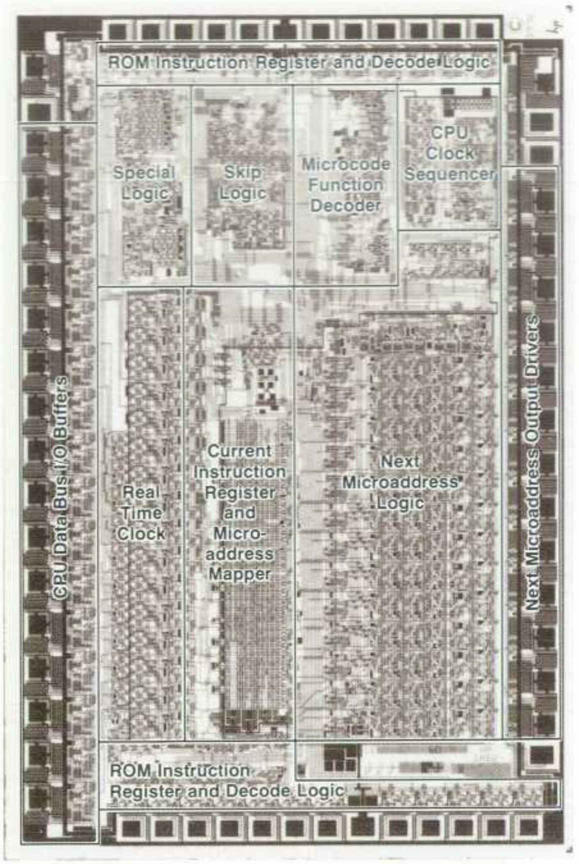

ROM Instruction Register and Decode Logic
Special Logic
Skip Logic
Microcode Function Decoder
CPU Clock Sequencer
CPU Data Bus I/O Buffers
Real Time Clock
Current Instruction Register and Microaddress Mapper
Next Microaddress Logic
Next Microaddress Output Drivers
ROM Instruction Register and Decode Logic

Fig. 5. *PCU chip.*

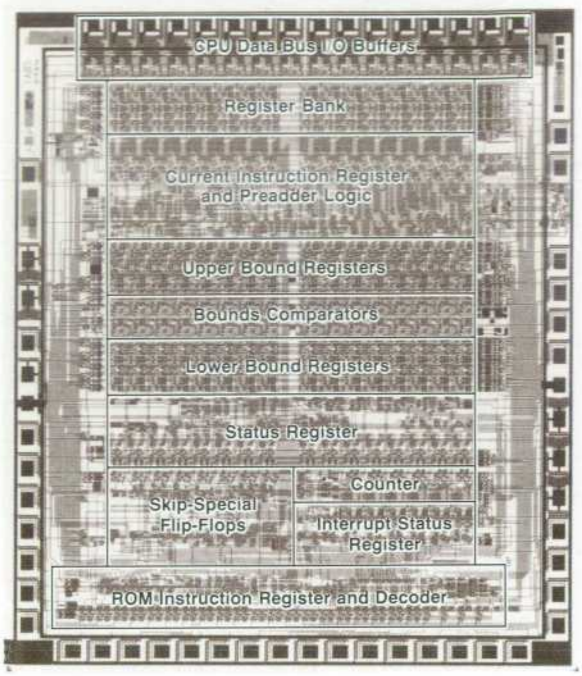


Fig. 6. *RASS chip.*

controller board and two or more memory array boards. The controller handles the IMB handshake and the error detection and correction of memory data words. Together with each 16-bit data word stored in memory array boards, there are six extra check bits; they enable the controller to correct all single-bit errors and detect all double and/or odd errors. The history of error activities is logged in the controller automatically to provide useful service information.

CPU Operation

The timing of CPU operations is controlled by the CPU clock generated by the PCU chip. The rising edge of this clock loads the new microinstruction into the ROM instruction registers, and the decoding begins immediately. A and B operands are selected and sent to A and B registers on the RALU chip. When the CPU clock changes state, the direction of the time-multiplexed CPU data bus is turned around, and the result from the ALU is sent to its designated destination. The fetching of the next microinstruction is done in a pipelined fashion: the PCU sends out the next ROM address while the current microinstruction is being processed. This overlap eliminates waiting for the next microinstruction.

A 32-bit microinstruction is interpreted as one of five different formats, depending on its function specification (Fig. 8). For example, one of the formats divides the 32 bits into seven control fields: three fields specify the operand pair and storage registers, two fields specify arithmetic/shift options, another field indicates conditional skips, and the last field sets the status of control flip-flops.

The HP 300 processor can be viewed as a three-address machine. However, not all formats provide this three-address capability. Jump address and ROM constants are included at the expense of other fields.

The microprocessor design is closely tailored to the HP 300 architecture. As an example, we can look at part of the STORE instruction, which pops the top element of a data stack into a memory location. This instruction specifies one of three possible base registers, a five-to-eight-bit displace-

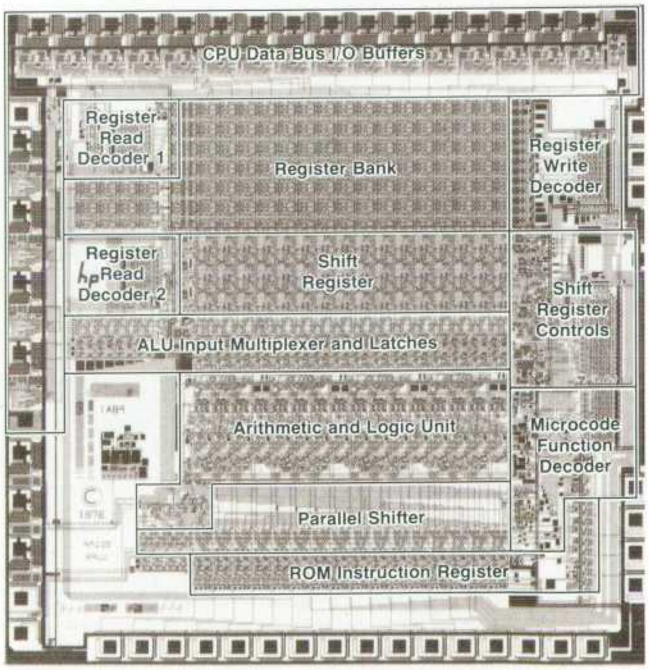


Fig. 7. *RALU chip.*

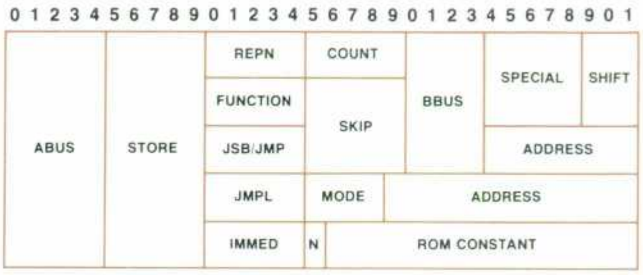


Fig. 8. *HP 300 microinstructions are 32 bits wide and come in five formats.*

ment, and an indexing option. The microprogram to emulate a STORE is shown below:

ABUS	BBUS	FUNCTION	LABEL	STORE	SPECIAL	SKIP
BASE	PADD	JSB	STRI	MSPE		INDR
TOSA		XFER		BUSD	BWRQ	UBND
S		CAD		S	CLAT	NEXT

The first line computes the operand address by adding a base register, a displacement, and the conditional index. The RALU hardware handles the BASE option and chooses the S, DB, or Q register as the ABUS operand. The RASS extracts the correct displacement and adds the index register to it if needed, providing the result (PADD) as the BBUS operand. The sum of these two operands (JSB implies addition) is stored into three registers (MSPE): the memory address register on the BIC board, the effective address E register for bounds comparison on the RASS, and an internal RALU register for future access. If the instruction is an indirect store, the PCU does a subroutine jump (JSB) to microroutine STRI and saves the return address in a hardware subroutine stack.

The second line transfers the top element in the data stack (TOSA) to the memory data register (BUSD) and writes it out to memory (BWRQ). The address in E is checked (UBND) against proper base and limit registers in the RASS. A violation would result in an immediate microprocessor trap before the data is written into memory.

To finish the STORE routine, the last line decrements the stack pointer(s) and pops the top element in the internal register stack (CLAT)—the PCU hardware simply declares current top-of-stack register TOSA invalid and the next-to-top register TOSB becomes the new TOSA. The end of the instruction is marked by NEXT, which causes the hardware to enter the instruction fetch phase.

Acknowledgments

Many people contributed to the design of the HP 300. We would like to acknowledge the design engineers who contributed their skills and foresight to the development of the HP 300 hardware:

CPU

Buck Gee, Chuck Glenn, Ron Guffin, Fred Ware, Jim Tseng, Woody Woodward, David Rabinowitz, Kent Andres, Lee Britton, Dom Chirieleison, Marcelo Siero, John Carlson, John Watkins, Tony Hunt, Jean Lundeen.

Memory

Ed Holland, Bill Stenzel, Bruce Eisenhard, Rich Hammons, Carson Kan, David Lu.

IDS

Al Knoll, Bernie Stewart, Norm Marschke, Dave Delano, Tom Gilbert, Hal Sampson, Janelle Bedke, Peter Rosenbladt, Jim Basiji.

Power Supply

Jim Mathios, Dick Ollins.

I/O Subsystems

Andre Schwager, Gordon Matheson, Bob Berliner, Ed Giaimo, Collin Park, Dean Lindsay, Lang Lok, Steve Sun, Jack Elward.

Diagnostic and Support Tools

Jim Lewis, Russ Scadina, Jim Holl, Dave Delano, Carson Kan, Joanie Banks-Hunt, Roger Ruhnow, Robert Paull.

Industrial and Product Design

Roger Lee, Scott Stillinger, Dale Dell'Ario, Beth Blomenkamp.

Technical Support and Engineering Services

Aida Silva, Darlene Harrel, Linda Pipkin, Corri Mooney, Ron Villata, Nellie Monsees, Bob Jones, Chuck Habib, Vince Napoli and staff.

Arndt B. Bergh

The HP 300 system architecture is the most recent in a long line of responsibilities at HP for Arne Bergh, who came to HP in 1956. His projects have included instruments, magnetic devices, memories, computers and the hardware design of the HP 3000. He's named as inventor on six patents and is a member of IEEE. A native of Canada, he received his AB degree in chemistry from St. Olaf College in 1947 and his MS degree in physics from the University of Minnesota in 1950. In his off-hours, Arne, a resident of Los Altos Hills, California, can be found honing his skill at sailboat racing.

Kenyon C. Y. Mei

Kenyon Mei joined HP in 1973 to work on HP 3000 hardware. He soon transferred to a new group investigating LSI computers, which eventually evolved into the HP 300. His responsibilities included CPU hardware and LSI chip set designs, and he is named as inventor on a patent on multiple-function logic gates that came out of his chip design efforts. He received his BS and MS degrees from the University of California at Davis in 1967 and 1968, and his PhD from Stanford University in 1974—all in electrical engineering. He enjoys part-time teaching and beginning tennis.

A Computer Input/Output System Based on the HP Interface Bus

by W. Gordon Matheson

THE BASIC INPUT/OUTPUT BUS of the HP 300 is the HP Interface Bus, or HP-IB, HP's implementation of IEEE Standard 488 and identical ANSI standard MC1.1. The integrated display system (IDS), the flexible disc drive, the system disc drives, the printers, and other devices interface to the system via the HP-IB. However, the HP-IB is not the only I/O bus allowed on the HP 300. The computer hardware supports up to fifteen I/O channels of several different classes. An HP 300 I/O channel is interfaced to the system by a channel controller board installed on the intermodule bus (IMB) to provide interrupt capability, DMA (direct memory access), channel program management facilities, and special protocol translation between the computer system and devices on the I/O bus. For brevity, channel controllers will be called "channels" in this article.

I/O Channel Characteristics

An I/O channel contains up to 16 read/write registers. It also responds to various I/O commands issued over the IMB for such purposes as processing interrupts and channel program service requests, channel identification, and initialization. The I/O system structure supports eight devices per channel. Each channel may incorporate a DMA facility for performing data transfers between main memory and peripheral devices without central processing unit (CPU) intervention. Priority for memory access is based on physical proximity to the CPU. Many I/O channels may conduct DMA transfers concurrently on the IMB, interleaving their memory cycles with the CPU on a priority demand basis.

At initial release the HP 300 has only two channel types, the 31262A General I/O Channel (GIC) for HP-IB interfacing, and the 31264A Asynchronous Data Communication Channel (ADCC) for management of up to eight RS-232 data communication links.

Device Reference Table

Each of the eight devices on an I/O channel has a four-word entry in reserved memory. The first word contains the address of the next channel instruction to be executed. The second word contains the memory address of a special interrupt parameter area, called the channel program variable area (CPVA). The third word is the label of the code segment to be executed when the device interrupts the CPU. The fourth word is used to coordinate and maintain the activity status of programmed I/O operations. The area of memory containing this information for all I/O devices is called the device reference table, or DRT (see Fig. 1).

I/O Operations

There are three types of I/O operations: direct I/O, programmed I/O, and interrupt processing.

Direct I/O operations are done with a set of I/O-oriented CPU instructions. They allow a privileged-mode programmer to read and write channel registers, perform special diagnostic operations, affect the ability of channels to request interrupts, reset and initialize the channels, and identify which channels are present on the IMB.

Interrupt processing is done within a multilevel priority structure. Channels are individually enabled and disabled for accessing a common interrupt request line by bits within a mask word broadcast to all channels simultaneously. This is done by the CPU to disable interrupts from lower-priority channels when it decides to service an interrupt from a particular channel. Interrupt request priority is determined strictly by channel number. The standard order is for lower

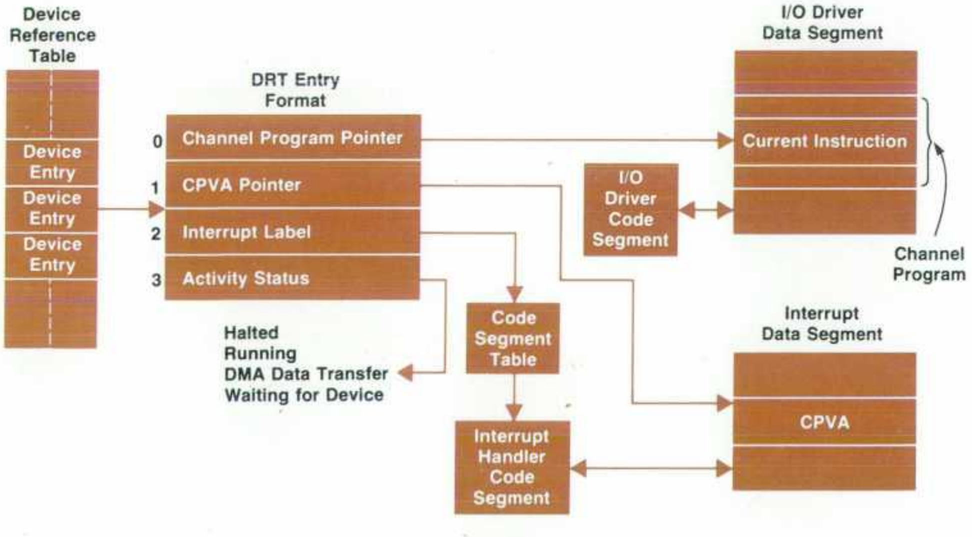


Fig. 1. *HP 300 hardware input/output system memory organization. Every I/O device has an entry in the device reference table (DRT). The channel program variable area (CPVA) contains interrupt parameters.*

channel and device numbers to have higher priority. However, each channel determines the priority of its devices and may provide means for the programmer to disable devices individually.

When the CPU is interrupted, it queries the highest-priority requesting channel to obtain the device number to be serviced. Then the CPU saves the old software environment on the stack and establishes a new software environment on the interrupt control stack in the interrupt code segment indicated by the device's DRT entry.

Programmed I/O is a means of accomplishing I/O operations through execution of special I/O programs that are initiated by software but executed by a special I/O processor or by the CPU in a special I/O processor mode. Programmed I/O is provided to allow complex I/O operations to proceed in parallel with software execution, and to remove the burden of detailed management of low-level I/O bus protocols from software.

Channel Programs

A channel instruction set is defined for the GIC as part of the basic system capabilities. This is used to create I/O-oriented channel programs, which are not executable by software (they are generally contained within a data segment). The channel instruction set is designed around a management framework for the HP-IB that provides interleaved service on several devices concurrently and independently. Many channel instructions are in high-level forms (buffer transfers, pauses, halts, computed jumps, etc.), with the details of HP-IB command structure and protocol being handled by a channel program processor.

Fig. 2 illustrates the logical operation of the I/O system. Channel program execution is initiated when a SIOP machine instruction is executed, specifying a selected channel, device, and starting address for the channel program. When the channel program terminates, a CPU interrupt will be requested for that device, with parameters indicating the reason for the interrupt stored in the CPVA. Executing the SIOP instruction on a channel that does not execute its own channel programs causes the channel to request channel program service on one of two CSRQ switch-selected lines on the IMB. CSRQ1 goes to the CPU. CSRQ2 is provided so that it may be possible in the future to design a separate channel program processor (CPP) to relieve the CPU of its channel program processing load. Channels that execute their own channel programs may do so simultaneously with CPU operations, and do not use a CSRQ line.

When the CPU sees a request on CSRQ1, it enters a special CPP mode of operation to execute the channel program for the device without disturbing the software environment. When the device's channel program is terminated or temporarily suspended pending a device request, the CPU resumes execution of software instructions until the next request for channel program service. The CPU will usually be in CPP mode a small fraction of the time.

The CPU checks for a number of errors during channel program execution, and may abort the program without affecting operations with other devices. Some detectable errors are: invalid instructions, illegal use of some instruction features, HP-IB lockup, and DMA memory errors. Each

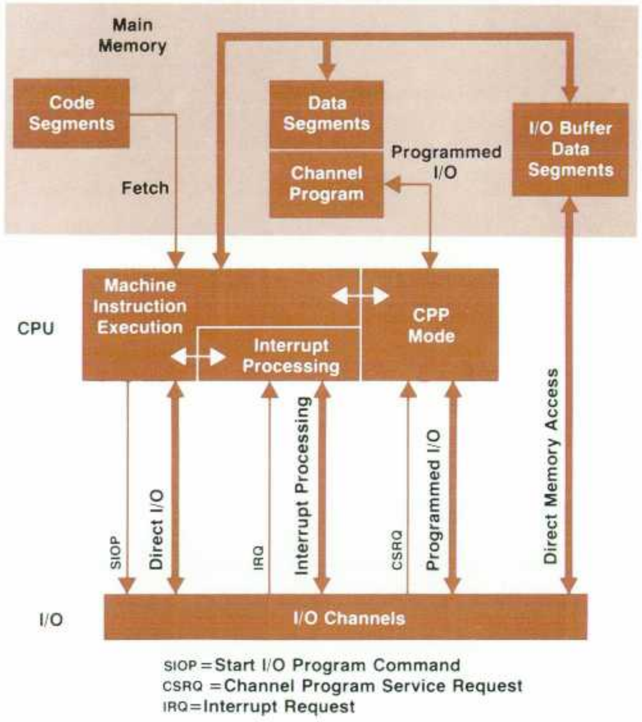


Fig. 2. *HP 300 hardware input/output system logical operation. Channel programs for input and output operations are written in a special instruction set and executed either by the I/O channels themselves or by the CPU in a channel program processor (CPP) mode.*

error is reported by interrupt with a special code in a reserved CPVA word. DMA error messages include the suspected offending memory address.

I/O Channel Types

There are three classes of I/O channels allowed by the HP 300 hardware I/O system, although not all have been developed.

GIC-Type Channels: This class of channel works with the channel program service procedures of the CPU. It translates all device requests into channel program service requests. This type of channel must look very similar to the 31262A GIC in register format, and it must appear to attach to an HP-IB.

Software-Controlled Channels: It is possible to have a channel that relies on software instead of programmed I/O to perform all operations. Such a channel translates device service requests into CPU interrupt requests.

Program-Interpreting Channels (PIC): It is possible for a channel to be designed to execute its own unique channel instruction set. Such a channel may use a programmed I/O approach similar to that of GIC-type channels, or it may use request and response queues for communication, executing on table structures instead of single-channel instructions.

System Identification Provisions

To facilitate system autoverification and autoconfiguration, identification features were added to all levels of the I/O system. First, there is an I/O instruction that identifies all used channel numbers on the IMB. Second, each channel

A Small, Low-Cost 12-Megabyte Fixed Disc Drive*

by Richard L. Smith

Early in the conceptual phase of the HP 300 project, the need for a resident low-cost mass memory was recognized. While many existing disc drives would have adequately fulfilled the needs of a system memory, none met the requirements of available space, power, reliability, and capacity necessary for the small integrated system concept of the HP 300. Thus, the design goals for this memory were: smallest possible size, 12-megabyte capacity, low cost, high data reliability, high performance, and compatability with the mounting, cooling, power, RFI, and HP-IB requirements of the HP 300 System. Fig. 1 is a photograph of the disc drive used in the HP 300.

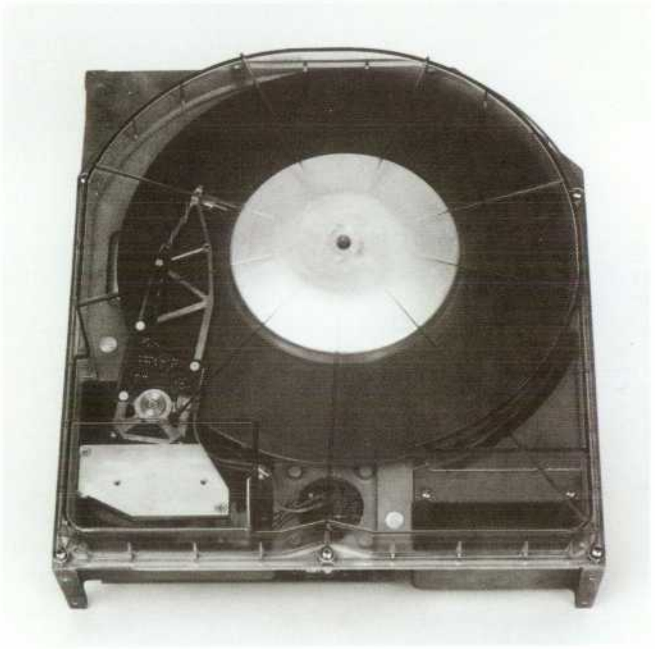


Fig. 1. *This 12-megabyte fixed disc provides built-in mass memory for the HP 300 Computer.*

The size goal dictated the use of a rotary (rather than linear) actuator, brushless dc (rather than ac) motor, and a single platter. The capacity goal required that both sides of the platter be used for data and that linear and radial densities be maximized. The cost goal indicated that a high degree of tooling would be needed and that a maximum use of microprocessor firmware rather than discrete circuitry was necessary.

The high data reliability could be met in one of two conventional ways: using MFM encoding and error correcting circuitry, or using a self-clocking code such as double-frequency FM. We opted for the lower capacity but highly reliable FM encoding technique and therefore saved the substantial cost of error correcting circuitry.

To use a lower-cost rotary actuator, it is necessary to use Winchester-type heads and media. The Winchester technology was pioneered by IBM Corporation. The heads are designed to take off and land from a lubricated disc surface and fly about 0.5 μm from the disc, which is rotating at 3000 r/min. Because of this extreme proximity of the head to the disc surface, cleanness is essential. The drive is manufactured in a class-100 clean environment and sealed from outside contamination. Internal air is continuously purged by recirculating it through a 0.3-μm filter.

To meet the capacity and performance goals, a technique of positioning the read/write heads was developed that uses the data surface directly as a reference, rather than a separate reference surface. Data is organized on a disc surface into concentric circles called tracks and blocks of data called sectors. Between sectors there normally are gaps with nothing recorded in them. In our intersector gaps we place information for locating the track center on a sampled basis, and track identification addresses in Gray code format so that we can update our position when seeking a new address (see Fig. 2).

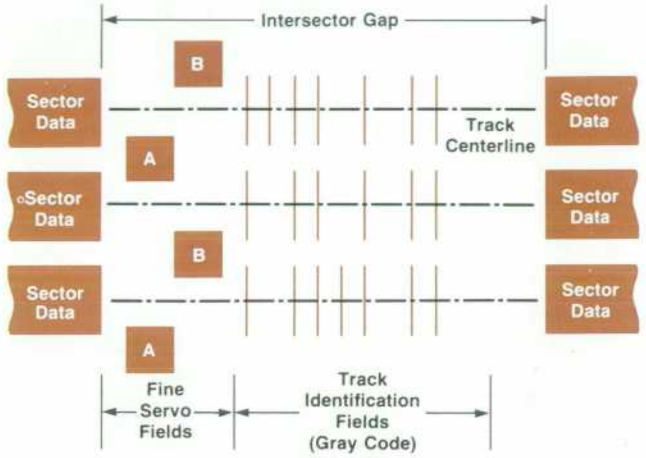


Fig. 2. *Information recorded in gaps between data records on the disc makes it possible to use the data surface as a reference for positioning the read/write heads.*

When a head is on track it reads equal amounts of A and B fine servo fields. This information can be decoded on a sampled basis, once per sector, and used for fine servo positioning.

When a head is traversing the disc surface in a seek mode, it samples the Gray code track addresses to update its position and

*Editor's note: A more complete article on this new state-of-the-art disc drive is planned for a future issue.

Richard L. Smith

Rich Smith, section manager in charge of mass memory development, received his BS degree in electronics and physics from the University of San Francisco in 1962. He joined HP in 1970 with experience as a magnetic recording specialist and read/write and servo designer. He was project manager for the 2644A terminal and the 7910 disc system, and designed parts of the 7970, 3960, and 3955 tape recorders. A native of San Mateo, California, Rich served as president of the Board of Education in San Jose, California while living there. The Smith family, which includes five children, now lives in Boise, Idaho where Rich's leisure time is spent camping, fishing, stamp collecting and working with hi-fi equipment.

servo on a real-time basis. To reduce the circuitry required for the seek algorithm, a resident microprocessor is used. When a new address is commanded, the microprocessor computes the distance to be covered and generates an acceleration signal based on the formula $s = \frac{1}{2}at^2$. As the actuator moves the head toward the desired location, the microprocessor monitors the track addresses and corrects the acceleration term as required.

Since a reference surface is not used, there is no clock track available and therefore spindle speed control is essential. The spindle hub includes teeth that are sensed optically. The edge of each tooth is used to determine a radial line across the disc that represents the beginning of each sector. The speed of each tooth is measured and used in a phase-lock servo comparator to keep the spindle speed constant. Additional circuitry monitors all internal voltages and the speed control so that data is written only when the drive is ready.

To maximize data transfer rate and signal-to-noise ratio, read and write amplifiers are mounted with the heads on the actuator arms. It was decided that a single read/write head per disc surface, rather than the conventional two heads per surface, was a better choice for a low-cost objective.

The built-in microprocessor allows the controller to be resident with the drive and is powerful enough to permit the inclusion of 56 separate self-test features including reading and writing to an unused track. Self-test is automatically initiated on power-up or may be commanded locally or remotely. Results of the self-test are displayed by lights on the controller board. A 256-byte sector buffer is included as part of the controller to allow buffered reads and writes to accommodate differing data transfer rates.

has a configuration register that identifies the channel class, specific type, and capabilities. On the HP-IB, the HP printers and discs, the integrated display system, and some other peripherals respond to a unique identification protocol whereby the system may determine attached device types and addresses without user intervention.

31262A General I/O Channel

The GIC contains registers and a DMA facility for interfacing the HP-IB to the IMB. It is designed to be operated using channel programs, so device requests on the HP-IB cause only channel program service requests and not interrupt requests. Interrupts are requested only by command of the CPP or CPU when required in the course of programmed I/O. Eight registers on the GIC are associated with management of the HP-IB, and are contained in the PHI chip, an HP propietary SOS/MOS LSI component.[1] The PHI chip provides all the interface functions of the HP-IB and performs the detailed handshaking and control sequences, keeping these details of HP-IB operation transparent to the CPU or CPP. DMA logic can be invoked to transfer data bytes between main memory and the HP-IB at up to one million bytes per second. The GIC has timeout logic to detect lockups or inordinate delays in HP-IB or DMA operation.

Channel Instruction Set

The HP 300 channel instruction set executed by the CPU is as follows:

READ, WRITE: Transfer bytes between an HP-IB device and main memory. After the transfer, the byte count and memory address residues are left in the instruction fields. Options:
1. Full record or subrecord (burst) transfers
2. Up to 15 data chain blocks
3. Data bucket/source using single memory word
4. Disable updating of residue byte count and address
5. Start/end on left or right byte of memory word.

DEVICE-SPECIFIED JUMP: Uses a byte of status from the device to do a table lookup for the address of the next channel instruction to execute.

IDENTIFY: Performs a unique HP-IB protocol sequence that returns an identification code from the HP-IB device.

CLEAR: Sends an HP-IB Selected Device Clear command to the HP-IB device to reset it.

COMMAND HP-IB: Sends up to eight HP-IB commands of the programmer's selection.

EXECUTE DMA: Performs initiation and termination bookkeeping for a DMA transfer without any HP-IB protocol.

RELATIVE JUMP: Branches unconditionally to continue channel program execution at a new address.

WAIT: Suspends the channel program until the device requests service on the HP-IB.

WRITE RELATIVE IMMEDIATE: Writes a literal into a specified word of the channel program.

There are also instructions for reading, writing, and modifying channel registers. The instruction set provides access to almost every capability of the HP-IB.

Acknowledgments

Development of a hardware I/O system requires diverse efforts in investigation, design, verification, characterization, and support. Almost two dozen engineering people have participated in such efforts from conception of the HP 300 I/O system until its first release to manufacturing. Preliminary investigation and proposals for the HP 300 hardware I/O system were made by Jim Basiji, Daryl Knoblock, and Al Knoll. Later definition and planning was done by Andre Schwager, George Clark, and Bob Berliner. I/O-related control programs for the CPU were developed by Arndt Bergh and Collin Park. Dean Lindsay helped develop the 31262A General I/O Channel. The 31264A Asynchronous Data Communication Channel was developed by George Clark and Collin Park. Jim Lewis and Tony Hunt

W. Gordon Matheson

Gordon Matheson helped design the HP 300 I/O systems, designed the general I/O channel (GIC), and wrote CPU firmwave for channel program execution. With HP since 1973, he's also contributed to the design of the HP 1000 Computer's CPU, memory protect, and DMA, and has written microprograms for the HP 1000 extended instruction set and front panel. Gordon attended the University of California at Berkeley, graduating in 1973 with a BSEE degree. A native of Glendora, California, he now lives in Santa Clara, California. He has two sons and a daughter.

were responsible for development of many diagnostic programs. Extensive system simulation work was done by Dan Jackson, Lang Lok, Dave Rabinowitz, Steve Sun, and Jim Tseng. I/O system integration and characterization were the responsibility of Jack Elward, Lang Lok, Collin Park, Steve Sun, Bill Stenzel, Majid Majidian, Roger Ruhnow, Barbara Gee, Rooshabh Varaiya, Joanie Banks-Hunt, Mike Lee, and Denise Pitsch. I/O system and peripheral diagnostics run on a special diagnostic utility system developed by Jim Holl, Russ Scadina, Bob Berliner, and Carson Kan.

Reference

1. J.W. Figueroa, "PHI, the HP-IB Interface Chip," Hewlett-Packard Journal, July 1978.

An Innovative Programming and Operating Console

by Alfred F. Knoll and Norman D. Marschke

THE HP 300's integrated display system (IDS) is a microcontroller-based alphanumeric keyboard and display system optimized for use as the system console and program development station in the HP 300. Full use of the powerful programming, display management, and editing features of the HP 300 dictates this somewhat unconventional, specialized programming and operating console. As an integral part of the HP 300 instead of a remote general-purpose terminal, the IDS shares the HP Interface Bus with the built-in fixed and flexible discs. Although it may seem unusual to have a programming console on the same interface bus as the computer mass storage devices, it is precisely this arrangement that gives the IDS its unusual features. By allowing the IDS the same communication capabilities as the discs, a very close interactive relationship with the CPU is made possible. Taking advantage of this association is the key to the IDS features.

The Window Concept

Perhaps the most revolutionary feature of the IDS is the idea of being able to view and manipulate portions of a number of independent display files that coexist simultaneously on a single CRT. Conventional terminals operate in an essentially serial fashion: the items displayed are sorted by entry sequence, so that the last item displayed is the most recent entry. By contrast, the IDS display is sorted by ultimate item organization.

This element of visual fidelity adds a new facet to the classical concept of interactive programming. Since multiple interactions can be sorted on the display by their environment rather than by entry sequence, the HP 300 need not follow the conventions imposed by ordinary terminals.

A window can be considered a viewport into a file, instead of a copy of a file. Using this concept, changes or additions to the display via the keyboard are made simultaneously to the actual file with the results immediately visible to the user.

Windows are implemented on the display by dividing the screen into rectangular subsets of the 24-row-by-80-column display. These areas are delineated by dotted lines called borders. The dotted lines occupy the space between adjacent characters, so the number of displayable characters is not reduced by the presence of these window borders.

Each window can be considered a separate display almost as if it were an independent terminal. The information in a window can be edited or scrolled both horizontally and vertically without altering either the contents or the position of the other windows on the screen. The capability to store more than one environment (or window set) locally and modify any eligible window, whether it is currently active or not, provides previously unattainable file manipulation power.

The Softkey Concept

The versatility of window-oriented display management is further enhanced by the softkeys. The eight softkey switches add a new dimension to task selection and user interaction. Program-definable labeling of these keys permits the creation of powerful, easy-to-use application programs not possible with conventional terminals. The labels defining the functions of the softkeys are displayed in a window next to the softkeys along the right side of the CRT.

Several important capabilities are provided by the seemingly simple dynamic labeling of these special-function keys. The first and perhaps most obvious is that of allowing the user to quickly select from a menu of options just what to do next. Suitable relabeling allows the selection of one of 512 items using only three keystrokes.

The next friendly feature realized by the softkeys is syntactical independence. For normal mortals, errors in system command creation and entry can be annoying, frustrating, and potentially disastrous. Use of the softkeys eliminates spelling and punctuation errors from the task selection process.

Yet another important feature is the simple fact that the possible next action options are displayed for the user's selection. One doesn't have to remember all the commands

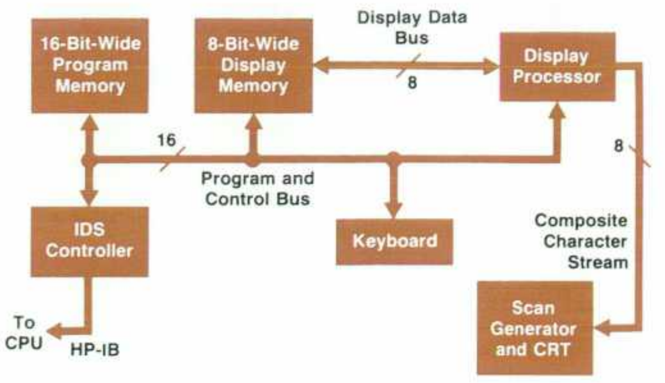


Fig. 1. *Dual bus structure enables close interaction between the HP 300 CPU, the integrated display system (IDS) controller, and the HP 300 display processor.*

or refer to a command summary or manual because the next set of menu items is displayed on the CRT. The problem of communicating one's desires to the AMIGO operating system is thereby vastly simplified.

Window Implementation

Implementation of the window-oriented display management features of the IDS requires very close interaction between the HP 300 CPU, the IDS controller (an MC² microcontroller[1]), and the HP 300 display processor. To enable this close association the IDS uses a dual bus structure (see Fig. 1). A 16-bit program and control bus is provided for the execution of the MC² firmware that handles CPU communications and controls the display processor. A separate eight-bit display data bus is used by the display processor for creating the composite character stream necessary to refresh the CRT. Functional separation of the display refresh and display editing processes gives the controller the bandwith necessary to support window management.

The display processor hardware incorporates two 80-character row buffers that ensure uninterrupted CRT refreshing. While one buffer is being filled with the composite character information for the next row of the display (characters, video enhancements, character set selection, and window border information), the other buffer is supplying character stream information to the scan generator for the row currently being displayed. When the row is completely displayed the buffers are functionally exchanged and assembling of the next row can begin. The operation of the scan generator is similar to that of the HP 2640A Terminal.[2]

The Display Program

The technique of creating the composite character stream was developed especially for the IDS. The characters are displayed on the screen as a set of 24 contiguous rows. Unfortunately this organization is not the most advantageous storage arrangement. To facilitate the editing necessary to provide both horizontal and vertical partial screen scrolling and the environment switching that is fundamental to the HP 300, a program and data access algorithm was devised for managing the display memory.

The display program is created by the IDS controller in response to window requirements from the user. This program resides in the display memory along with the text that it puts on the CRT. The display processor accesses this program and creates the composite character stream in response to the program parameters.

The display program is made up of two parts, the instruction table and the text (see Fig. 2). The active instruction table is a sequence of four-byte instructions sufficient to define all the rows in the currently viewed windows on the screen. The text is a series of linked buffers in the same memory, containing ASCII character codes and various embedded display control information bytes. Each row in each window is defined by one four-byte instruction and its associated text buffer list.

The table is accessed as a series of four-byte instructions. Each instruction refers to an individual row of text in a single window. For a vertically partitioned screen the first instruction refers to the first row of text in the first or upper left window area. The next instruction refers to the first row of text in the window immediately adjacent to the right border of the first window. For the simplest single-window

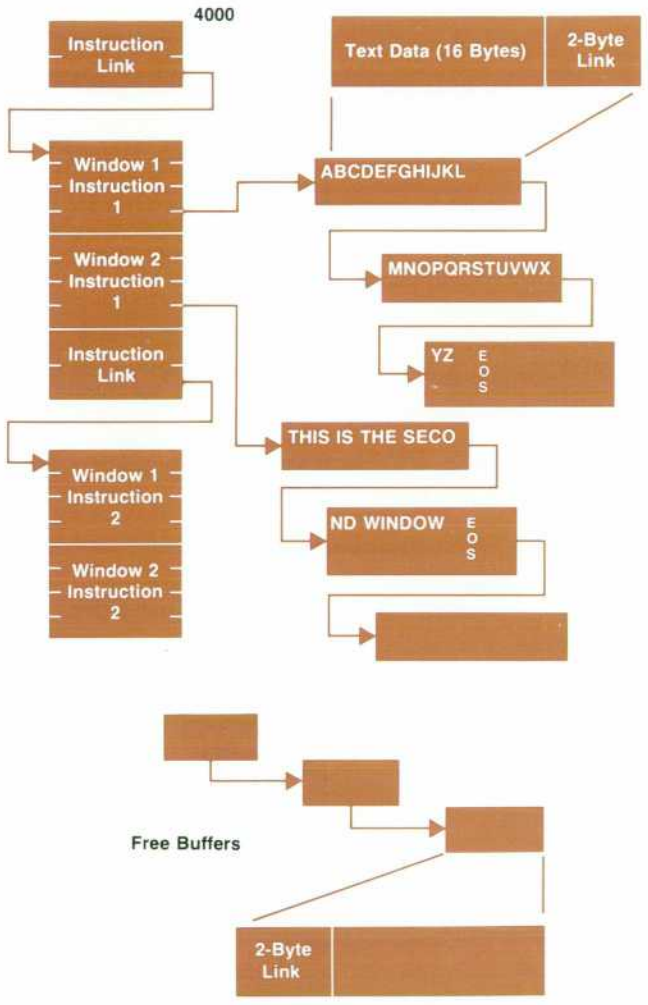


Fig. 2. *IDS controller creates a display program in response to the user's window requirements. The program, organized as shown here, consists of a series of four-byte instructions and text. Text is stored as a series of linked 18-byte buffers.*

Bit Number

8	9	10	11	12	13	14	15
0	Window Width						
Right Border	Bottom Border	Character Set		Half Bright	Under-line	Inverse Video	Blink
Last	X	Data Address MSBs					
Data Address LSBs							

Fig. 3. *IDS display instruction format.*

screen, 24 display instructions are required, one for each row in the window. The total number of instructions required for a particular display depends upon the number of rows that are partitioned vertically. If six of the 24 rows are partitioned between two vertical windows, these six rows require $2\times6=12$ instructions in addition to the instructions required for the remaining 18 rows.

The format of the display instruction is shown in Fig. 3. The first byte in the four-byte instruction contains a zero in bit location 8 (bits 0-7 are not used by display instructions). This indicates to the display processor that the four bytes are instructions rather than an instruction link. The next seven bits contain the window width measured in character positions. The second byte in the display instruction is the initial video enhancement byte for this row in this window. The third byte in the display instruction is used in conjunction with the fourth byte to point to the text data that is to be displayed in the window. Bit 8 of the third byte set to 1 indicates that this is the last window in the row.

Text Data Format

The text data is usually stored (by memory management convention) as a series of linked buffers with each containing 18 bytes. As far as the display processor is concerned, the buffers can be any length. However, the memory management firmware uses 18 bytes as the standard length. The buffer format is 16 data bytes followed by a two-byte data link.

8	9	10	11	12	13	14	15
0	ASCII Character Code						
1	0	Data Link—Address Most Significant Byte					
1	1	0	Not Used				
1	1	1	0	Half Bright	Under-line	Inverse Video	Blink
1	1	1	1	0	(Ignored by Display Processor)		
1	1	1	1	1	0	Character Set	
EOS (End of String)							
1	1	1	1	1	1	1	0

Fig. 4. *Linkage and enhancement information is embedded in the text to be displayed. To distinguish between characters and linkages, a Huffman-coded set of data byte formats is used.*

Since linkage and enhancement information is embedded in the text, a Huffman-coded set of data byte formats is used to distinguish between characters and linkages (see Fig. 4). ASCII characters that represent text to be displayed have a 0 in the most significant bit followed by the seven ASCII bits. A data link used for linking the lists together is identified by a 1 in the MSB followed by a 0 in the next bit. The succeeding bits become the MSBs of the pointer. The next byte in the buffer is used as the LSBs of the pointer, much the same as an instruction link.

It is important to note that the various types of data bytes may be arranged in any order within the data list. The leading-ones coding makes them positionally independent, unlike the instruction bytes, which must be sequential within each four-byte instruction. This feature allows character set selection and video enhancement changes to be mixed easily within the same row of text.

Sample Display Program

Fig. 5 shows the display screen produced by the sample display program diagrammed in Fig. 2.

The sample program is designed to partition the upper six rows of the display area into two windows. The first window is 30 characters wide and the second is 50 characters wide. The first row of the first window contains the alphabet, while the first row of the second window contains the words THIS IS THE SECOND WINDOW. All other rows in the first two windows are blank. The first row of the last window (18 lines of 80 character positions each) contains the text THIS IS THE THIRD WINDOW, while the remaining rows are blank.

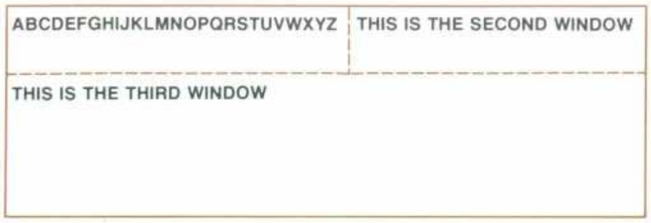
Recalling the previous discussion, it is apparent that $2\times6+18=30$ display instructions are required for this sample program.

The display processor begins each frame at the top of the instruction table by accessing the four-byte instruction for row 1 in window 1. Then, as directed by the data address in the instruction, it accesses the linked data buffers until the window width is satisfied. For row 1 of window 1, the data list is 16 characters of the alphabet followed by a link to the remainder of the alphabet. When the end-of-string (EOS) flag is encountered in the data list, the row is blank-filled to the end of window 1. The processor then proceeds back to the instruction table for row 1 of window 2 followed by the data list for window 2.

The second and subsequent rows of windows 1 and 2 are blank, so the addresses in the instructions for these rows/windows point directly to the EOS flag.

In a similar manner, each row of the display is processed with one or more sequences of an instruction fetch followed by a data list. After the last row on the screen, the processor resets the instruction address to the top of the table and starts all over again (every 1/60th or 1/50th second) for refresh of the next frame.

Using this program technique, vertical or horizontal scrolling requires only the alteration of the data pointers in the instruction table. Environment switching can be accomplished by creating a set of window information that is not referenced in the current instruction table, then merely inserting an instruction link in the table to point to the new window instructions.


ABCDEFGHIJKLMNOPQRSTUVWXYZ | THIS IS THE SECOND WINDOW

THIS IS THE THIRD WINDOW

Fig. 5. *Display resulting from the program of Fig. 2.*

Multilingual Capabilities

The problem of satisfying international keyboard and character graphic standards is easily solved in the IDS by a combination of hardware and firmware.

The display processor block contains space for up to three alternate character graphic ROM sets. Each of the sets contains an identity code that is read and stored by the controller during the power-on sequence. This code allows the controller to configure the alternate character set keys on the keyboard, and provides a linkage mechanism for the multilingual features.

The IDS keyboard is self-scanning, and interrupts the controller whenever the state of any key changes. This frees the controller from the tedious task of scanning the keyboard and provides rollover and autorepeat capability. A seven-bit keystation number is returned whenever the keyboard interrupt is serviced. The fact that this number has no direct relation to any ASCII character or code is the key to multilingual configurations.

Fig. 6 depicts the transformation algorithm used to implement the multilingual feature. The ROM ID obtained at power-on from the extension graphic ROM contains two information fields: the map select field selects a base set map corresponding to the base set graphics ROM, and the table select field is used to point to one of eight extension tables.

To process a keystroke, the selected base set map is addressed with the keystation number and the upper- or lower-case shift bit. The byte thus addressed is one of three types: local control, base set ASCII, or indirect vector.

If the key is a local control key, such as SHIFT or DEL ENTRY, a local control code is returned to the controller program and no character graphic is produced. If the key meaning is invariant regardless of the language option, a base set ASCII character is generated and used as an input to the base set character graphic ROM to produce the character on the CRT.

If the key meaning is dependent upon the language option and hence the extension graphic ROM, an indirect vector is returned. Six bits of this vector are used as an offset to access a particular location in the extension table

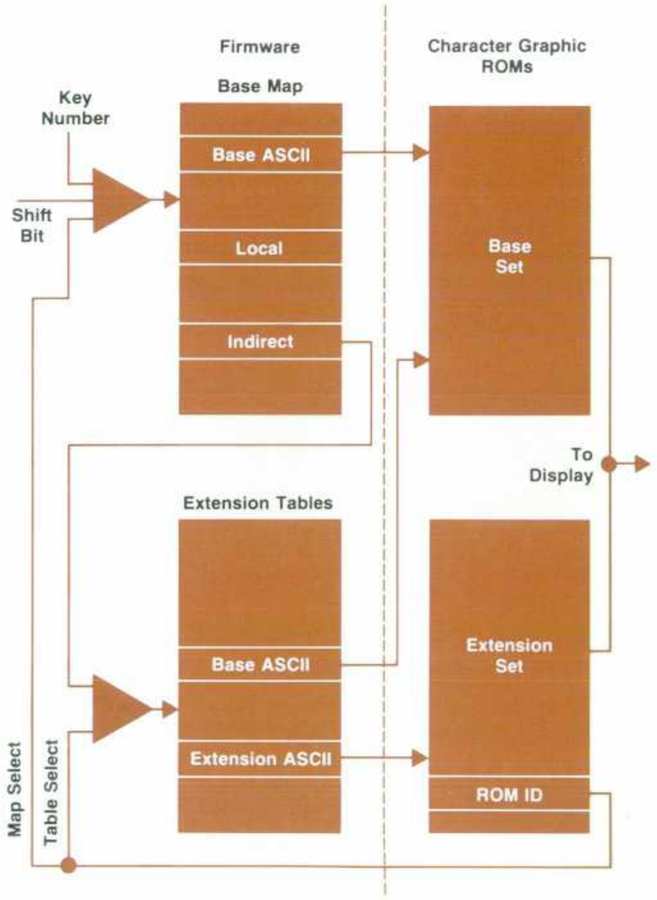
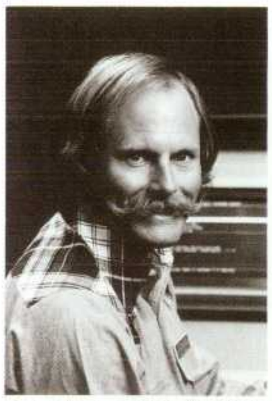
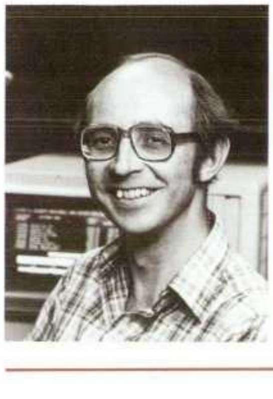


Fig. 6. *Multilingual transformation algorithm selects either a control code or a character to be displayed depending on the key that was pressed and the language being used.*



Alfred F. Knoll

Al Knoll was project leader and system designer for the HP 300 integrated display system. After receiving his MSEE degree from Santa Clara University in 1970, he worked for two years in radar system simulator design before joining HP. Al designed the 12889A Hardwired Serial Interface presently used as the high-speed communications link in HP 1000 to HP 3000 distributed systems. With the HP 300 from its beginnings, he followed the IDS design to manufacturing as the production engineer. An avid skiing enthusiast, Al lives in Los Altos, California. Ethnic cookery, small boat sailing, racquetball, and bicycle touring are some of his spare time interests.



Norman D. Marschke

Norm Marschke is presently involved with HP 300 mainframe power supply, safety, and EMC design. His past contributions to HP include serving as project leader for the 5470A Fast Fourier Processor and contributing to the development of counters, the 5400A Multichannel Pulse Height Analyzer, a digital shaker control system, and the IDS and power supply of the HP 300. He joined HP in 1964, after receiving his BSEE degree (1963) and MSEE degree (1964) from the University of Michigan.

specified by the value of the table select field of the ROM ID. The contents of this location is a byte containing an ASCII code and an extension ROM select bit. If the desired character is not part of the base set, the select bit enables the extension graphic ROM and the ASCII code specifies the character. If, on the other hand, the particular character is not changed under the language option in question, the base set is selected, and as before, the ASCII code specifies the character.

The maps and extension tables for all the standard European keyboard configurations are contained in a single ROM that is standard with the IDS. Configuring the IDS to support a different keyboard arrangement is merely a matter of placing the proper extension ROM in its socket and replacing or rearranging a few keycaps.

Testability

One of the primary objectives of the IDS design was testability. The successful meeting of this objective revolved around the development of a comprehensive but simple self-test capability. This is achieved by dedicating part of the firmware to a set of self-test routines and providing a simple means of displaying test status.

The IDS self-test is invoked at power-on and takes approximately three seconds. An HP LED array mounted on the controller printed circuit assembly and visible from the rear of the HP 300 displays the test results in hexadecimal form. The test is designed to require no external signals other than power from the mainframe and thus establishes a measure of goodness for the IDS independent of the state of the remainder of the HP 300. The test is comprehensive enough to create about a 90% confidence level in the functional integrity of the IDS.

Acknowledgments

The uniqueness of the IDS is the result of the combined contributions of many outstanding people. Hal Sampson designed the IDS controller and several of the firmware development tools, and greatly influenced the overall system design. Bernie Stewart designed the keyboard and was ultimately responsible for the automated testing of all the circuit assemblies. Nellie Monsees fabricated the prototypes and diligently programmed the DTS-70 test system. The microcode for the many innovative IDS features is the result of design efforts by Tom Gilbert and Wing Chan. The self-test firmware and diagnostic software were created by Dave Delano. The entire project owes much of its success to the guidance of Peter Rosenbladt, presently marketing the HP 300 in Europe, and numerous other members of the HP 300 team who contributed user comments and suggestions.

Reference

1. B.E. Forbes, "Silicon-on-Sapphire Technology Produces High-Speed Single-Chip Processor," Hewlett-Packard Journal, April 1977.
2. J.C. Roy, "A High-Resolution Raster Scan Display," Hewlett-Packard Journal, June 1975.

AMIGO/300: A Friendly Operating System

by Ralph L. Carpenter

THE FIRST FEATURE most people notice when seeing an HP 300 in operation for the first time is the ease with which the man/machine gap may be bridged. For example, the operating system's user interface subsystem, the console handler, responds to natural-language commands, such as COPY FILE A to B. Most commands may be interrupted by the ATTN key on the integrated display system (IDS). When the ATTN key is pressed, execution of the command (in this case duplication of a file) continues, but two of the softkeys become labeled CANCEL COPY and ACTIVATE COPY. Cancellation of the copy results in the shutting down of the in-process duplication; activation simply results in the original display, that is, just the COPY command in the input window and the softkey labeled HELP.

This design of an improved man/machine interface, sometimes known as friendliness, has been carried throughout the design of the HP 300 operating system, AMIGO/300, resulting in modularity, maintainability, supportability, and ease of distribution. AMIGO/300 supports multiprogramming, multitasking, virtual memory, a large dictionary of commands, tools for synchronization of task execution, and transportability of programs. The purpose of this article is to present an overview of AMIGO/300's structure and functional modularity, and to give the reader more of an understanding of what makes the HP 300 tick. File management and terminal management, which are operating system services, were discussed last month and are not included here.

Command Interface

The IDS serves as the console for the HP 300. It has several novel features that are managed by the console handler portion of the operating system.

1. Softkeys. There are are eight keys along the righthand side of the screen. Labels showing the functions of the keys may be displayed on the screen.
2. Windows and borders. The IDS can accommodate several display structures on the screen at one time, separated by borders (series of tiny dots). One such window

is dedicated to softkey labels.

3. ATTN (attention) key. This key causes an immediate asynchronous interrupt.
4. System message light. This LED blinks whenever the system needs the operator's attention.
5. Scrolling capability. One window at a time may be shifted up or down and left or right to provide full visibility of the data attached to that window.

In addition to the softkey window, the console handler maintains the environment window, containing the generation name of the AMIGO/300 operating system along with the current domain and the current date and time, the input window, in which all typed commands and all command interaction are displayed, the error window, in which all interactive command errors are displayed, and one large display window that may optionally be divided in half. The display window contains scrollable output from previous console activity.

All console activity is logged, beginning with the welcome banner (system identification, command language version, and date of release), a list of device differences between the active configuration and that which the system was told to expect, the AMIGO/300 physical file name for the console being logged, the file name of the previous console log (only one previous log is retained), an indicator showing how many times the log has overflowed, and for each command that executes successfully, the date and time execution finished, along with the date and time of each job termination.

A job represents one execution of the user's application program or of an HP-supplied subsystem (like BASIC). The operator is allowed to run several concurrent copies of the same program by using the form RUN P AS J1. This causes program P to be run as a background job. If the operator uses the shorter form RUN P, program P is run as an interactive job. Ownership of the system console is automatically granted to the interactive job and must be taken from it via the ATTN key and specifically given to another job via the ACTIVATE softkey or command. A job may consist of several programs, because one program may load and start another program using the standard systems services known as program management. The only level at which a job is defined is the console handler level.

Command Language

The AMIGO/300 system command language was designed to be as friendly as possible while providing a powerful and useful tool for users. It has an easy-to-learn syntax and an English-like grammar, enabling the operator to form command sentences that have obvious meanings.

Imperative sentences consisting of a verb followed by an object form the basis of the language. Declarative sentences are used in some cases where appropriate. Some commands allow modifying clauses that supply optional parameters to the command interpreter. These modifying clauses may take various forms depending on the operator's wishes. Abbreviations of language keywords are allowed, and misspellings of language keywords are corrected for the operator (up to a point). The command interpreter is sensitive to common errors such as character transpositions and replications, and it can handle missing, incorrect, or extra characters depending on the length of the keyword and its context.

A command sentence is, in effect, a pattern of keywords and data items. Whenever a keyword is required in the pattern and the string entered as input does not match the pattern, an error condition exists. Fortunately, many errors of this type can be corrected dynamically. The abbreviation rules already discussed are an example, as are common typing errors such as replicated characters and simple letter transpositions. More severe errors such as missing or incorrect characters are correctable only in limited cases. Correction of this type of error depends heavily on the length of the input string and the context in which the keyword is required.

The AMIGO/300 operating system is the first HP computing system to employ sophisticated input correction techniques. Therefore, it is anticipated that users will require a certain amount of experience to feel comfortable with the system.

Several factors that are not obvious contribute to the reliability of the methods chosen to do dynamic error correction. First, the system is very context sensitive. It operates under the assumption that the user knows in general what he or she wants to do, even though the exact syntax of the command statement may not be known by the person entering the command. Thus only keywords legal in a given context are "candidates". This list is always much shorter than the list of all keywords known to the command interpreter. Second, before a correction is done, all candidates are examined and a candidate is chosen only if it is a unique choice that differs from the typed input by no more than an internally set threshold. Finally, the different sentence structures of the various commands, the distribution of keywords, and checks on the data entered add the same redundancy to the command language that are present in spoken communication. This further diminishes the possibility that an erroneous command input will be accepted even if an erroneous keyword substitution is made.

Example: The command verb DUPLICATE may be entered in any of the following ways.

DUP	substring abbreviation
DPLCT	abbreviation by vowel removal
DPULICATE	transposition
DUPLIKATE	wrong character

There are three general classes or types of errors reported by the AMIGO/300 command interpreter. These include errors in the command pattern input (called syntax errors), errors in the meaning of data in the command (called semantic errors), and system errors detected during the execution of the command.

Syntax errors are errors detected during the interpretation of the command input by the user and represent an uncorrectable failure to match the input with a valid command pattern. When this occurs, the cursor is positioned in the IDS input window at the symbol that caused the error, and a descriptive message is written to the IDS error window. These messages generally indicate what specific keyword or data item is required, so in many instances a person can learn the syntax of a command by interaction with the error handler. More general messages are reported when no command verb is found and when the list of legal options is too long to fit in the input window.

Semantic errors are errors detected during the execution of a command. These generally concern data that has been input as part of the command, and they deal with the meaning of the data. For example a PURGE FILE command may be entered with the syntactically correct file name F1(ME). The command may still fail because no file by that name exists or the user attempting the PURGE does not have access to the file. Semantic errors are reported in much the same manner as syntax errors, except that in some cases, the cursor is simply reset to the beginning of the command.

Finally, a command may fail during execution because of a system failure. For example a DUPLICATE could be terminated by an error in the I/O system, or because it is explicitly aborted by the user. Execution errors are also reported in the IDS error window.

Program Management

A program, as the word is commonly used, is a series of instructions telling a machine how to behave. On the HP 300 this form of program exists as a workspace, that being the set of files containing the source code, relocatable object code (both in unlinked, symbolic form and in a form that has had all its externals resolved), compiler listing, segmenter listing, linker listing, a file used by the symbolic debug facility for mapping code and data into their corresponding source line number and data symbols, and a file containing file equations for execution of the program in the particular machine environment. In AMIGO/300, the word "program" is most commonly used to describe the installed (loaded) machine-level representation of a workspace.

A program is, first of all, a member of a job. Remember, however, that the job concept exists only at the console handler level in AMIGO/300. A program may contain code that calls an AMIGO/300 service for loading and starting programs. These services are part of AMIGO/300's program management facility. Such programmatic loading/starting of a program (more specifically, a workspace is loaded and the corresponding program is started) creates a hierarchy of programs. Bottom-level programs may terminate, and that is that, but if a program terminates at a higher level in the program hierarchy, all of its descendant programs must be terminated also.

In addition to the program termination and abort capability, program management also provides for blocking and unblocking of subordinate programs, inquiring as to the status of a subordinate program, and retrieval of a parameter array supplied at program startup.

Task Management and Synchronization

The primary structure of an execution environment under AMIGO/300 is known as a task. These are the basic building blocks that make up programs, which in turn make up jobs. Initially, there exist only two tasks in each program environment, one known as the outer block task and the other known as the when task. The when task is used solely by the AMIGO/300 operating system and is not visible to the user; it is employed to support asynchronous (or without wait) completion of file and terminal input/output. The outer block task may create and start sibling tasks, but there exists no hierarchical relationship among tasks within a program environment. Each task has its own arithmetic and control stack, used for procedure linkage, parameter passing, and procedure-local storage. All tasks share the same set of code segments in the program environment. Any sibling task that is created or started is limited to those code segments that are in the workspace. Thus, an outer block task may only create or start a sibling task at a procedure that is within the workspace. In addition to this set of code segments, all tasks within a program share a global data storage area and a set of program-local data segments. Task management services include creation, initiation of execution, aborting, blocking and unblocking, alteration of a task's priority as well as other environmental parameters, and other services.

Tasks may communicate information to one another by a number of methods. The easiest to understand is the memory file communication technique. This technique is recommended for interprogram communication, when the sending task and the receiving task are in two separate program environments. A memory file is simply a system-owned (protected) data segment whose access is strictly controlled by the file management facility under stringent rules:

1. Only one program may read (receive) information from the file, although multiple programs may write (send) into it. Although multiple tasks in the receiving program have access to this file, it is left to the application programmer to synchronize the sequence of reads or to specify that all reads are to be performed by one task.
2. Queuing is first-in-first-out. A read request issued against an empty memory file causes the receiver to wait until a data element is sent by an active sender. Sending is done without delay, unless the memory file is full.

Another method of intertask communication, which is not permitted between programs, is via ID numbers and events. For each program, the system maintains a list of ID numbers, each of which is a positive integer. ID synchronization services are applied to this set of ID numbers. They allow the user to refer to system objects (tasks, etc.) without granting the user direct access to the corresponding control blocks. Since ID numbers are local to each program, synchronization cannot occur across program boundaries, except via files.

There are five primitive functions for intertask synchronization: signal, wait, wait-any, request, and release. These primitives can be applied to system-supplied events (e.g., wait for a write to the printer to complete), or they can be applied to programmatically reserved ID numbers. ID numbers reserved by programs are called user-created IDs, and their meanings are left to the program (for example, an ID might represent completion of a series of computations, with the result finally being stored into an area of storage that is commonly addressable by two different tasks). The wait-any primitive may be used where a number of asynchronous signals or multiple IDs are expected (e.g., multiterminal read operations). The request and release primitives are reserved for use on a very special type of ID, the resource semaphore.

In a program that includes multiple tasks, common data must be protected from conflicting access by several tasks.

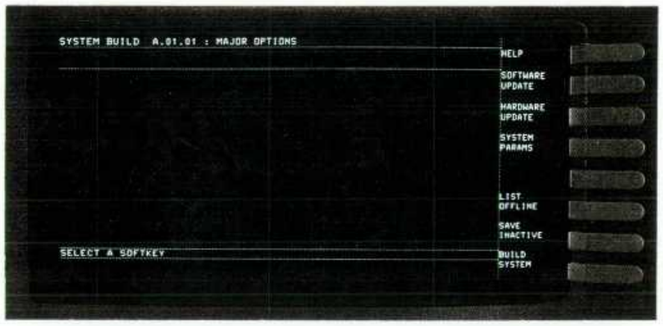
Configuring and Launching the AMIGO/300 System

by Donald M. Wise and James C. McCullough

SYSTEM BUILD is a privileged system program that provides the capabilities of creating, modifying, or deleting an HP 300 software configuration, a process sometimes referred to as system generation. It is invoked in the MANAGER domain by the command BUILD SYSTEM, and can be run concurrently with other programs. After completion of SYSTEM BUILD it is not necessary to stop the system immediately. The currently-running software configuration remains in effect until the next SYSTEM STARTUP (see below), at which time the newly built software configuration takes effect.

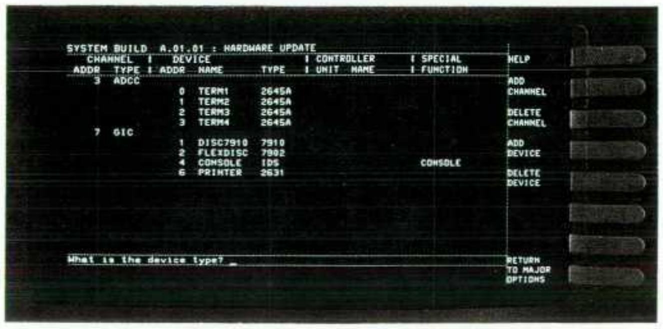
SYSTEM BUILD takes full advantage of the power of the integrated display system (IDS) by extensive use of windows and softkeys. Configuration options are displayed in menu-like fashion in the softkey window and can be randomly selected via the softkeys (see Figs. 1 and 2). Configuration parameters are specified by answering questions in the interactive window. There is no command syntax to be learned. A sequence of questions can be terminated by simply selecting another softkey instead of answering the question. The current configuration is displayed in the display window during specification of each option, and is then updated to reflect the specified change. Each user input is checked for errors, which are reported in inverse half-bright video in the error window. And of course the HELP facility is available via a softkey, with text entries keyed to the configuration parameter questions to reduce index searching.

SYSTEM BUILD is easy to use not only because of the power of the IDS, but also because of underlying design goals of minimizing user interaction and of using terminology that is familiar to the user. SYSTEM BUILD does not require the user to respecify the entire configuration each time. Instead, the user starts with an existing configuration and specifies only the changes that are to be made. User interaction is also simplified by allowing changes to be specified in terms already known. For example, to add the IMAGE/300 data base software, the user specifies the name IMAGE/300 and the appropriate volume label, but does not need to know the names of the data files, libraries, and workspaces that make it up. Or, to add a printer to the hardware configuration the user specifies its name (e.g., PRINTER1), device type (e.g., 2631) and its hardware location (e.g., channel 1, device 7), but does not need to know the name of the printer driver. An added benefit of such simple terminology is a reduced possibility of error.


```
SYSTEM BUILD  A.01.01 : MAJOR OPTIONS

SELECT A SOFTKEY
```

Fig. 1. *System generation on the HP 300 is accomplished with the help of* SYSTEM BUILD*, a privileged system program.*


```
SYSTEM BUILD  A.01.01 : HARDWARE UPDATE
  CHANNEL   I    DEVICE                 I CONTROLLER       I SPECIAL
ADDR  TYPE  I ADDR  NAME        TYPE    I UNIT  NAME       I FUNCTION
   3  ADCC
                0   TERM1       2645A
                1   TERM2       2645A
                2   TERM3       2645A
                3   TERM4       2645A
   7  GIC
                1   DISC7910    7910
                2   FLEXDISC    7902
                4   CONSOLE     IDS                         CONSOLE
                6   PRINTER     2631

What is the device type? _
```

Fig. 2. *Configuration options are selected by means of softkeys. Configuration parameters are specified by answering questions in the interactive window.*

The process of building a system involves three phases: initialization, specification, and build. In initialization the user selects the configuration that is to be modified, which can be the currently running configuration or an inactive configuration (a description of a configuration that was saved during a previous SYSTEM BUILD session). During the specification phase the user changes the starting configuration by adding or deleting software and hardware. At this time the modified configuration may be listed on the printer and can be saved as an inactive configuration for initialization during a subsequent SYSTEM BUILD session. This feature allows the user to stop SYSTEM BUILD and resume later without having to respecify changes already specified. Also during the specification phase, the configuration is checked for errors. The user is not allowed to proceed to the build phase until all such errors have been corrected.

The nature of the build phase depends on whether or not the entire configuration must be built. If the software configuration has changed, or if the starting configuration is an inactive configuration, then the entire configuration must be built. The user is prompted to mount the flexible discs containing system software. Then system code segments are linked together, system tables are built, and system programs (e.g., the BASIC compiler and SYSTEM BUILD) are prepared and linked to the new system. The entire process takes 45 to 60 minutes or more, depending upon the optional software, list device, system disc, system memory size, and other activity on the system. However, if the starting configuration is the currently running configuration and if the software configuration has not changed, then only the system tables must be rebuilt, a process requiring less than five minutes.

Another major design goal of SYSTEM BUILD is the ability to recover from unexpected hardware and software errors. For example, if an error is detected by SYSTEM BUILD (such as an I/O error while writing to the printer) the user is informed of the nature of the error and the file or device involved. The user can then retry the request. If SYSTEM BUILD is unexpectedly terminated (e.g., by a user command or a power failure), the currently running system configuration remains in effect. The user can then rerun SYSTEM BUILD and attempt the build without having to restore a backup of the currently running configuration. Once the new configuration is built, the currently running configuration is saved as a backup. If the new configuration cannot be started for some reason (e.g., insufficient memory because of large system parameter values) then the backup system is automatically started. The user can then run SYSTEM BUILD, correct the error, and rebuild the new configuration. Once the new configuration has been started, the backup can be purged via SYSTEM BUILD to conserve disc space.

Among the more advanced features of SYSTEM BUILD is its cross-configuration capability. A configuration can be specified that is completely different from the currently running configuration. This

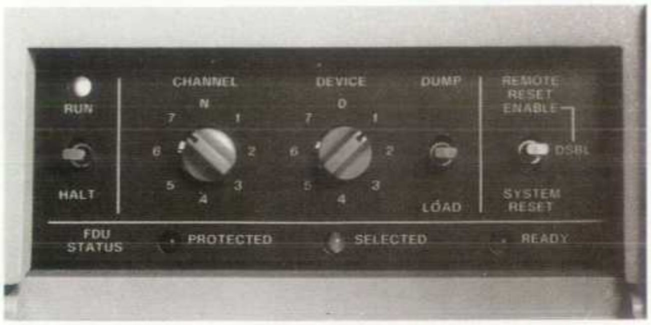


Fig. 3. *HP 300 control panel is used when starting the system.*

configuration can then be built on a foreign system volume, and this volume can be transported to another machine with a compatible hardware configuration and used as its system volume. Other advanced features of SYSTEM BUILD include tools that are useful in operating system development but are not normally used by HP customers.

Starting the System

SYSTEM STARTUP consists of two privileged programs responsible for launching the AMIGO/300 operating system. SYSTEM STARTUP gains control via the LOAD or POWER ON sequence as a privileged stand-alone program and completes the launch task as a privileged program with access to AMIGO/300 system services.

The initiation of SYSTEM STARTUP is accomplished by user manipulation of switches on the HP 300 control panel (see Fig. 3). First, the user dials the channel and device numbers of the system disc, then turns on the integrated system followed by the peripherals, and then presses HALT, RESET, and LOAD.

In keeping with the HP 300's friendliness and ease of use, several features simplify the task of starting up the AMIGO/300 operating system. SYSTEM STARTUP launches the operating system without user interaction. The only exception occurs when completion of a system dump is required after an abnormal system shutdown. Changes involving real memory size, the number of I/O channels, I/O channel mix, and number and status of I/O devices are detected by SYSTEM STARTUP and reported to the user. If SYSTEM STARTUP determines that it cannot successfully complete launching the currently configured system, it will attempt to launch the previously configured system. This is possible because SYSTEM BUILD always saves a backup system when a new system is generated.

Acknowledgments

To those who poured lots of time and energy into making the HP300 a successful product, thank you. Your efforts are very much appreciated. We would like to pay special tribute to the following persons for their outstanding performances in producing SYSTEM BUILD, SYSTEM STARTUP, and the SOFTWARE MANAGEMENT SYSTEM. Lee Lebowitz contributed to the development of SYSTEM BUILD. Jim Lewis developed SYSTEM STARTUP with Mike Hartstein contributing the SMART DUMP utility. Pat Trytten, Bob Ashford, and Rick Meyers were the developers of SOFTWARE MANAGEMENT, a means of integrating system software and preparing it for manufacturing.

James C. McCullough

Now section manager for AMIGO/300, Jim McCullough was project manager for the HP 300's SYSTEM BUILD and SYSTEM STARTUP utility programs. With HP since 1972, he's also written diagnostics, an I/O driver, and communications software. Born in England, Arkansas, Jim attended Arkansas A, M, and N College, graduating in 1962 with a BS degree in mathematics. During the next ten years he gained experience as a mathematician, a high school math instructor, and a computer programmer. He's married, has three children, and lives in San Jose, California, and his major interests are choral music and sports.

Donald M. Wise

Don Wise was project leader for the HP 300's SYSTEM BUILD utility program. With HP since 1974, he's designed several I/O drivers and a system generator for the DOS/MX operating system, and is now a project manager in the AMIGO/300 lab. Don attended Stanford University, where he was a member of the band and was in charge of the computer-designed card stunts for Stanford football games. He graduated in 1974 with a BS degree in mathematical sciences, and received his MSEE degree in 1975. A resident of Sunnyvale, California, a few miles from Redwood City, his birthplace, Don serves as a volunteer in a recreation program for the handicapped that is directed by his wife. He also enjoys racquetball, camping and hiking, bicycling, and his beer can collection.

For example, a program may have a list structure that is asynchronously accessed by several tasks. If one task begins to insert a new entry and has updated only some of the linkage pointers when a second task suddenly accesses the list, the second task will encounter incorrect and perhaps invalid pointers. If several tasks simultaneously try to insert new entries in the same place, then the result is likely to be a very hard-to-find bug. These problems can be avoided by use of resource semaphores. Each table, list, or similar entity is thought of as a resource that is owned by at most one task at a time and is controlled by a resource semaphore. Before accessing any such shared entity, a task must request temporary ownership of the resource. When ownership is granted, then the task can freely manipulate the associated data structure, since any other tasks that subsequently request this resource will be delayed. When the owning task has finished the operation, it must release ownership. This allows another task to be granted ownership and continue its execution. Thus, all tasks that cooperate by issuing the appropriate request and release requests will have mutually exclusive access to the resource.

Virtual Memory Management

The AMIGO/300 memory management services provide virtual storage capability to programs operating on the HP 300. This feature is invisible to the programs, that is, no planned segmentation or overlay structure need be invoked by a program. There is a limit on the code segment size (currently 16K words), but one routine may call another without having knowledge about the segmentation. Arrays may be very large (the current limit of total directly addressable array storage is about two megawords for an application program), thus reducing the need to resort to temporary file storage of intermediate results. Also available through the memory manager is the ability to alter the size of the arithmetic/control stack, dynamic allocation and deallocation of arrays, the ability to alter the size of an array, and the ability to inquire into the parameters controlling the bounds on segment sizes.

Other invisible facilities provided by the memory manager to support the virtual environment include disc file block transfers (used by file management), buffer allocation and freeze-down/unfreeze (used by file management and terminal management), migration of segments from memory to disc (used by the scheduler), and miscellaneous other internal services.

Virtual memory configurations include both random access memory and bulk disc memory. Variable-length code and data segments are subject to migration between memory and disc according to the following rules:

1. Individual segments migrate in (i.e., from disc to memory) when demanded either by a microcode-induced trap or by an explicit migration request from some software module.
2. Groups of segments migrate in before execution of a task provided that the segments were previously migrated out (i.e., from memory to disc storage).
3. Groups of segments migrate out whenever a task encounters a relatively long suspension.
4. Individual segments migrate out whenever they impede an inbound migration request. Candidates for outward migration are carefully selected on the basis of the owning task's priority, the state and age of the segment, and the content of the segment.

Storage space for an array may reside in one of three locations: in the global region, on the arithmetic/control stack, or in one or more data segments. Which region is used depends upon the language in which the program is written. In the HP 300 system language, array space is allocated in the global region when the declaration

type ARRAY name (range)

appears in the program's main procedure, or the declaration

OWN type ARRAY name (range)

appears in a procedure, and range is a relatively small number. Array space is allocated on the stack when the declaration

type ARRAY name (range)

appears in a procedure, and range is a relatively small number. Data segment space is allocated for the array when the declaration

(OWN) type ARRAY name (*) or (range)

where range is a relatively large number, appears anywhere in the program. The preceding discussion applies to array allocation through the declarative capability of the HP 300 system language. In addition to declarative allocation, the programmer may employ the ALLOCATE statement for dynamic size adjustments.

The arithmetic/control stack is used for storage of procedure-local variables, intermediate computational results, parameters for called procedures, and stack markers (i.e., return code segment number, program register, and old value of local-storage stack pointer). The stack may be expanded (but not shrunk) automatically by the AMIGO/300 operating system upon a trap condition known as stack overflow. Once the stack reaches its maximum size, AMIGO/300 is not able to grow the stack any further. A stack overflow trap occurring when the stack is at its maximum size causes the program to be aborted.

A memory management service is provided to disable automatic stack expansion. Another service may be invoked to shrink the stack. Controlled expansion/shrinkage of the stack may be done at any opportune time in the progress of the application or subsystem (whereas automatic expansion may introduce a thrashing condition).

Two important points should be considered in regard to programs intended to execute in a multiprogramming virtual memory environment. First, one can never predict the execution-time program mixture. Therefore, few assumptions should be made about guaranteed performance. Second, the amount of random-access memory available to the task is always dependent upon instantaneous machine-load considerations, such as priority preemption, buffer freeze-downs, and so on.

Timer Services

An application programmer can use the timer services in many ways. For example:

1. To retrieve today's date and/or the current time, for printing dated (volatile) reports, for logging of transactions, for computing elapsed time, for noting the time to an interactive application program user, and so on.
2. To retrieve the job date (i.e., the date parameter specified in the RUN command) for preprinting checks, invoices, and so on.
3. To perform a conversion from one date form to another, for example from Julian to month/day or vice versa.
4. To add a date (Julian form only is supplied) and a constant for sales forecasting reports and the like.
5. To begin a watchdog timer at the same time that another request of possibly indefinite duration is initiated. This guarantees that, by use of the wait-any synchronization service, there will be some reasonable limit to the delay. For example, the system might be programmed to prompt an interactive terminal user that input is requested, and find, by timer expiration, that there is no one at the terminal.

Trap Handling

A series of arithmetic computations is usually done without regard to the data being operated on. Thus an underflow, overflow, divide by zero, or similar condition may occur. Even an experienced programmer may accidentally incur bounds violation traps—for example, by execution of an instruction that references an uninitialized address parameter, or by indexing beyond the end of an array.

When a trap of this type occurs, the AMIGO/300 trap handler does not abort the program, as is usual, but instead transfers control to a user-supplied or library routine. In a user-supplied routine, files and data bases might be brought to a known state before the program is aborted. A library routine might be more intricately tied into an error reporting package (e.g., the AMIGO/300 formatter).

In the case of a bounds violation, the trap handler looks to see if the symbolic debug package is configured into the program, and if so, invokes the symbolic debug package's trap handling routine to interact with the operator and to report where in the program the violation occurred.

Internal Interrupt Handler

The internal interrupt handler takes care of every microcode-induced trap. Many of these imply that the HP 300 hardware is responding erroneously, or that there is some inconsistency within AMIGO/300 itself, in which case the system is brought to a graceful shutdown (as graceful as possible under the circumstances). However, most of the traps fielded by the internal interrupt handler are normal and are expected to occur. For example, code and data segments under AMIGO/300 are normally not locked into memory, so they may be absent when needed. In such a circumstance, the microcode understands that the desired segment is not resident in memory, and that AMIGO/300 will take care of making the segment resident before continuing execution of the task that incurred the absence trap. The internal interrupt handler does not resolve the issue of making the segment resident and rescheduling the task for later execution. Instead, it calls upon the memory manager and the scheduler to do so.

Other expected events that are processed by the internal interrupt handler include:

1. Bounds violation. The trap handler is invoked if the code that incurred the violation was other than system code.
2. Arithmetic overflow, underflow, invalid operand (e.g., divide by zero), invalid character. These also result in invocation of the trap handler if encountered in program code.
3. IPL, or cold-load. This trap invokes the AMIGO/300 startup program.
4. Power-fail and power-on (to the central processing unit, or CPU). Currently, AMIGO/300 does minimal recovery.
5. Timer (CPU clock) trap. Transfer of control is passed to the kernel (see next section) for handling of the timer trap.
6. Debug instruction executed. Control is passed to the system debug facility.

The internal interrupt handler also fields many unexpected traps, most of which result in a soft crash of the AMIGO/300 system.

The Kernel of AMIGO/300

The kernel of AMIGO/300 consists of the following services, which are a basis for the multitasking environment:

1. The dispatcher, which is responsible for selection of one of the tasks in the system to become active
2. Allocation of control blocks that define tasks, events, semaphores, I/O request elements, and generalized control structures
3. Handling of timer traps, maintenance of the watchdog timer queue, and initiation of timer requests
4. Setting and retrieving the date and the time of day
5. Starting, stopping, retrieving, and initializing a task's CPU execution timer
6. Recovery from power failure
7. Blocking and unblocking tasks
8. Synchronization primitives (wait and post)
9. I/O initiation and completion
10. Generalized queue-handling routines.

This kernel consists of two major facilities. One is known as the control program, but since it bears no resemblance to a program as formally defined in AMIGO/300, it is simply referred to as CP. The other kernel facility is the I/O system, consisting of a driver interface in two forms: the initiator interface, known as the I/O request service, and the completor/continuator interface, known as the dispatcher. Device drivers are not included in the kernel I/O system, but they must, of course, conform to certain behavior patterns to be qualifiable.

Each task is bound to a list of completed (i.e., posted) events for that task. Of course, each posted event must be configured as owned by an existing task before invocation of the post primitive. The wait primitive examines this list of events, waiting for the appearance on the list of the particular event specified. The wait-any primitive takes the first-found completed event (if there is one) and processes that one. When the wait (or wait-any) primitive is unable to locate the event in question, an instruction is executed that causes the dispatcher to begin execution, and suspends the previously executing task. A special type of event, nicknamed the when event, is capable of vectoring execution of the task to a preconfigured procedure entry point. The when event is processed whenever a wait-any primitive is requested by a task to which the event has been configured and the event is already complete (i.e., posted), or upon completion of such an event for which a task is in the wait-any state.

Resource semaphore locking/unlocking primitive services provide a means of traffic flow control, such that data structures, etc. that are going to be revised or investigated by one task can be protected from investigation or revision by another task, where both tasks may be executing the same code. The current AMIGO/300 implementation permits only one task to lock any semaphore. An attempt to lock by another task results in the second task being blocked (that is, taken off the dispatcher's active task list) until the owner relinquishes control via an unlock, at which time the dormant task becomes the owner and is unblocked. Measurements will reveal whether this simplistic approach to semaphore queueing is adequate or not. If not, the queuing algorithm can be changed without revision of higher-level AMIGO/300 code.

Scheduling and Dispatching

As mentioned above, the dispatcher is part of the kernel of AMIGO/300. Its primary duty is to put tasks into execution, but it also responds to I/O drivers' interrupt handlers requesting that completion processing be performed. In the latter case, the dispatcher simply calls the driver's completor section, having found the entry label in a fixed table. I/O completion processing can be preempted by further interrupts, but only one completor may run at a time.

Dispatching of tasks is a process of searching the dispatcher's active task queue for the first unblocked task. Since tasks are installed by the scheduler in priority order, the first-found task will be of highest precedence. Another function related to the dispatching of tasks is the termination of a task's execution. This occurs whenever the task requests a wait service or when the task's remaining-CPU-time counter goes to zero. Once the dispatcher or the wait primitive determines that the task currently executing should cease execution, the scheduler is invoked to arbitrate. If there is nothing for the task to do, that is, if the wait primitive invoked the scheduler, the task is automatically blocked, and may be removed from the dispatcher's queue should the event being awaited be a long-term event. Generally, all events are long-term except the completion of a disc transfer. If entry into the scheduler was caused by a task's CPU time running out, the scheduler must assess the state of the task relative to other tasks in the scheduler's queue, a superset of the dispatcher's queue. If need be, the current task is blocked by the scheduler. It may also be dequeued from the dispatcher's list. When an awaited event occurs, the event is linked into the waiting task's complete list by the post primitive. If the list was previously empty, entry to the scheduler is again made. This time, the scheduler may find the task in a state that permits it to run if unblocked, so the scheduler unblocks the task. On the other hand, the task may be off the dispatcher's runable task queue, in which case the scheduler must see whether or not the task's memory resources have been consumed, and if so, must ultimately stage the task's working set back into memory. We now arrive at the portion of the scheduler that is tightly coupled to the memory manager.

Whenever an absence trap (previously discussed) occurs, the memory manager looks for free memory of adequate size to hold the demanded segment. If it cannot find sufficient free memory, it begins to try different means of making it available. If all fails, it invokes the scheduler for a policy decision: Is the task in question of sufficient precedence to warrant the displacement of another task, or is it not? This decision involves potential queue manipulation, and very possibly a task switch (i.e., dispatch) to a different task.

System Debug Facility

The system debug facility has been more instrumental in wringing out mistakes in the rest of AMIGO/300 than any other feature. With it, one can insert, clear, or list breakpoints (permanent, temporary, or counting), display code and data segment contents, print a trace of the control stack, display global or stack storage locations, display absolute memory locations, display register contents, and modify code and data segment contents. By knowing where the kernel keeps its data structures, the systems programmer can look at a task's current state, find out whether an event is complete or not, locate driver storage and external device states, and obtain many other pieces of vital information.

Acknowledgments

There were many contributors to both the architecture of the HP 300 and to the AMIGO/300 operating system. The author wishes to take this opportunity to acknowledge the primary software architecture designers, Arne Bergh, John Couch, and Bill Williams, who together derived the software environment on which AMIGO/300 was eventually to execute. One could not ask for a better team of architects than Arne, with his extensive microcode and hardware design experience, John, with his extensive background in compiler design, and Bill, with his extensive operating systems knowledge. The AMIGO/300 operating system modules represent the concerted and diligent efforts of Jim Avera, Chris Crump, Bill Dalton, Steve Gadol, Bill Haccou, John Hawkes, Ken Macy, Frank Mendoza, Bill O'Shaughnessy, Bill Parrish, Gary Wermuth, and Bill Williams. The console handler was developed by Ken Macy, Steve Gadol and Chris Crump, the program manager and trap handler by Gary Wermuth, the task manager, synchronization services, and scheduler by Jim Avera, the memory manager by Bill Haccou, Jim Avera, and John Hawkes, the internal interrupt handler, system debug, microcode-assisted trace facility, and simulation environment by Bill Williams, the timer services by Frank Mendoza, Bill Dalton, and Jim Avera, the kernel (dispatcher, control program and I/O system) by Bill Dalton, Frank Mendoza, Bill O'Shaughnessy, and Bill Parrish.

Ralph L. Carpenter

Ralph Carpenter came to HP in 1972 from Louisiana State University (BSME, 1967), and the University of California at Berkeley (MSME, 1968). Project manager for the AMIGO/300 operating system for four and a half years, he has also been project manager for another operating system, technical support manager for 2100 software, and technical support engineer for 2000C/F BASIC Timeshare Systems. Before joining HP, Ralph was a systems programmer, and he is named as an inventor on a software patent for a text editing system. Now a resident of San Jose, California, Ralph has two children. His interests include disco dancing, reading, listening to music, playing the guitar, playing snooker, and swimming.

A Multiple-Output Switching Power Supply for Computer Applications

Designed for computer mainframes, this OEM power supply is an economical solution for the HP 300's power requirements.

by Dilip A. Amin and Thane Kriegel

MOST COMPUTER SYSTEMS are powered by custom power supplies, designed for a specific application. There are reasons for this: tradition, size or shape constraints, and the fact that each new computer has its own sequencing or status reporting requirements. Recent advances in technology have made power supply development a major expense, so that it no longer makes sense to develop a new power supply for each new product.

The power supply used in the HP 300 is a standard, off-the-shelf power supply designed for computer mainframe applications. The HP Model 63312F four-output 550W modular switching power supply provides 5V at up to 50A, ±12V or ±15V at up to 10A and 40V at 1A.

Use of this commercially available unit was made possible by features incorporated in the 63312F and by partitioning the power system in such a way that the under and overvoltage shutdown and on-off sequencing circuits are not in the power supply, but in the system. The 63312F has two shutdown terminals that can be used to control the outputs for sequencing, undervoltage, and so on. In addition, a 15V bias source is provided by the 63312F to power the external system's supervisory circuits, even when the power supply outputs are shut down. The ±12V outputs of the 63312F can be changed to ±15V by simply shorting two terminals.

By incorporating these features in the power supply and by keeping the system supervisory circuits out of the 63312F, HP has been able to use this same standard off-the-

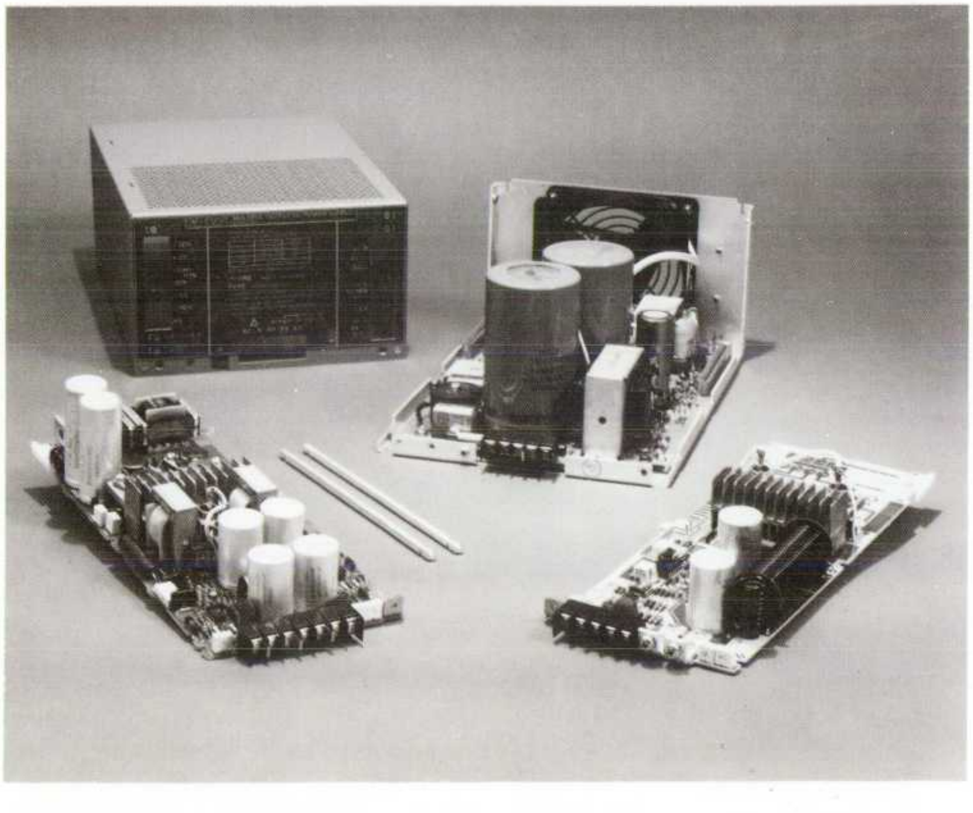


Fig. 1. *Three interlocking, functionally independent printed circuit boards contain all the circuits in the 63312F Multiple Output Switching Power Supply. The supply provides up to 550 watts at 5V, ±12V or ±15V, and up to 40V.*

shelf power supply in two other computers, the HP 3000 Model 33 and the HP 3000 Series III.*

As computers move out of the controlled environment of the computer room and into the office or factory, additional safety and electromagnetic interference (EMI) requirements are imposed. Since most of the impact of these requirements is on the power supply, using a standard power supply that meets worldwide safety and EMI requirements simplifies getting approval of the system.

The 63312F Power Supply is designed to meet many of the worldwide safety requirements. It is one of the few assemblies connected directly to the power line, and therefore is designed not to fail in such a way as to present a shock or fire hazard to the operator. In addition, the power supply was designed to minimize the amount of electromagnetic interference conducted back onto the power line.

The 63312F is packaged in a 203×292×127-mm enclosure (Fig. 1). The circuits are contained on three interlocking printed circuit boards. The circuit boards are functionally independent. The input ac line circuits are all on the motherboard, mounted horizontally in the bottom of the chassis. The EMI filter and safety isolation circuits are also on this assembly.

*Editor's note: Articles on these systems are planned for a future issue.

The two vertical plug-in boards are the output circuits. One board contains the 5V, 50A output circuits and the 40V, 1A output circuit. The other board contains the ±12V to ±15V output regulators. Forced air cooling keeps the internal temperature rise to only 4°C and improves the reliability of the power supply. Putting all the components including the 50A output circuits on printed circuit boards eliminates most hand wiring. This provides consistent performance and reliable low-cost wave-soldered connections.

Theory of Operation

Fig. 2 is a circuit diagram of the 63312F Power Supply. The ac power line voltage is rectified and filtered to provide an unregulated 300-volt dc source. A doubler/bridge configuration allows either 120 or 220 Vac input voltage operation. Line voltage selection is achieved by external straps on the input barrier strip. Thermistors R1 and R2 are provided to limit the inrush current necessary to charge the energy storage capacitors C1 and C2 when power is applied.

The 20-kHz inverter circuit converts the 300-volt dc source to a pulse-width-modulated (PWM), 150-volt-peak

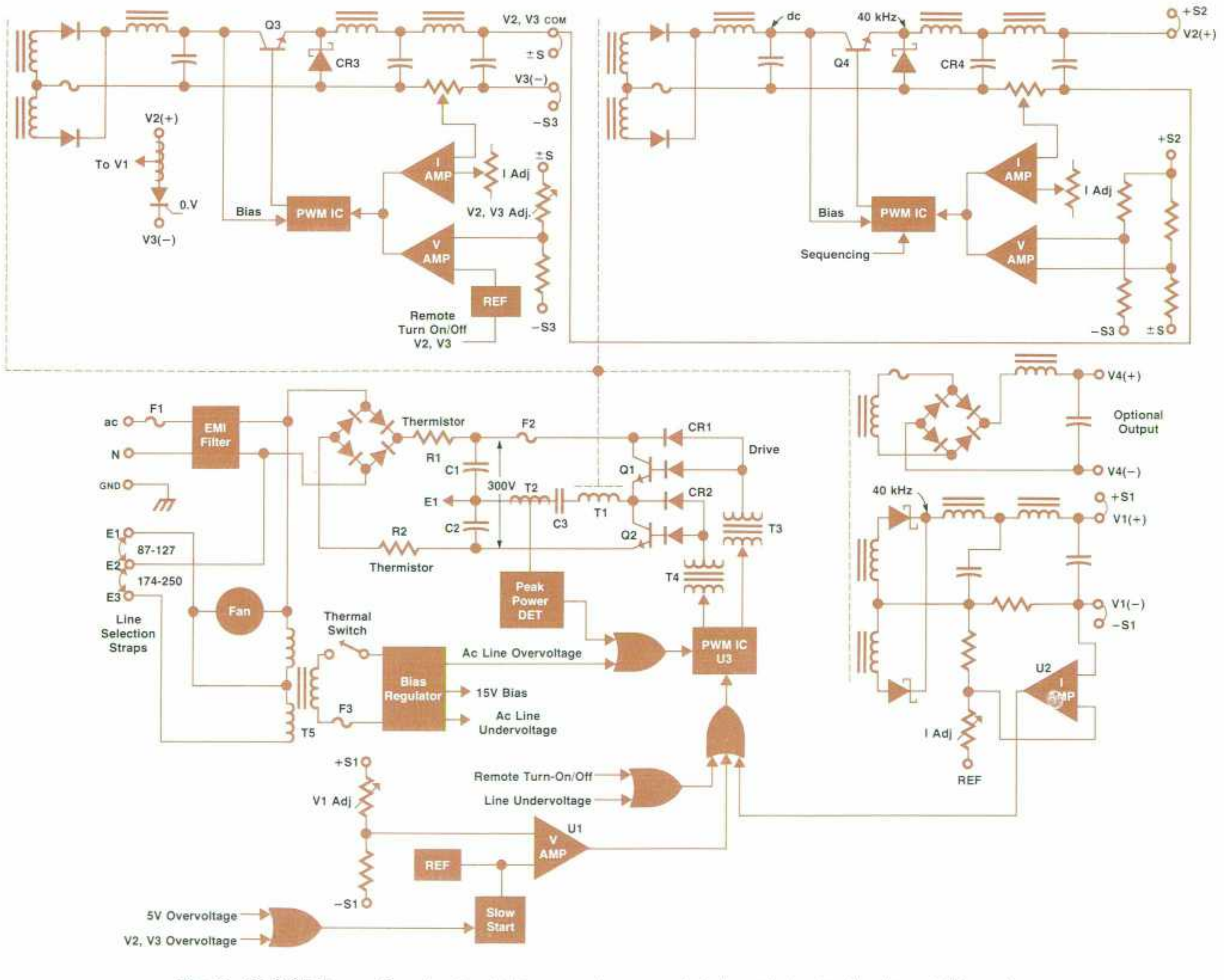


Fig. 2. *63312F Power Supply circuit diagram. Large-scale-integrated-circuit pulse-width modulators (PWM ICs) control the conduction periods of the inverter circuits.*

square wave. This circuit consists of flux-balancing capacitor C3, inverter transformer T1, switching transistors Q1 and Q2, fuse F2, and current transformer T2. Alternately switching transistors Q1 and Q2 on and off creates the 20-kHz PWM square wave. Diodes CR1 and CR2 keep transistors Q1 and Q2 in the active mode during the on part of the cycle and a negative base voltage is applied during the turn-off transition to hasten turn-off and minimize switching losses in Q1 and Q2.

Besides voltage transformation, inverter transformer T1 provides safety isolation between the primary and regulated outputs. The stepped-down secondary voltages of T1 are rectified and average-filtered to provide the five-volt output. The flux balancing capacitor, C3, prevents dc flux build-up on T1 caused by asymmetrical characteristics of Q1 and Q2. However, unbalanced flux can occur during transients, resulting in high magnetizing currents in T1. These currents are sensed by the current transformer, T2, and the appropriate correction signal is communicated to the control circuitry.

The five-volt output (V1) is controlled and regulated by voltage and current error amplifiers U1 and U2. The signals from the error amplifiers are fed to U3, a large-scale integrated-circuit pulse-width modulator (PWM IC), to control the conduction period of the main inverter. The PWM IC has min/max pulse width limiters and its output drives the inverter transistors Q1 and Q2 through the isolated drive transformers, T3 and T4.

A 60-Hz bias transformer, T5, provides regulated bias voltage to the control circuit and drive power to inverter transistors Q1 and Q2. In addition, bias is provided to power external circuitry for system monitoring and control. The fan is connected with the autotransformer primary of bias transformer T5 to accept either 120 or 220 Vac input power.

Auxiliary Regulators

Two additional regulated outputs, V2 and V3, are provided. These outputs derive their inputs from auxiliary windings on inverter transformer T1. These voltages are rectified, averaged-filtered, and regulated by a 40-kHz switching regulator in the continuous current mode. The turn-on/off circuit for these regulators is shown in Fig. 3.

The control circuits are similar to the main five-volt output control circuit. Voltage and current error amplifiers provide the control signals to PWM ICs to turn off the switching transistors. (A clock turns on the switches.) Flyback diodes maintain continuous current in the output filter chokes.

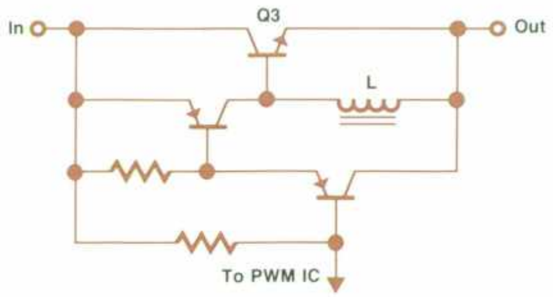


Fig. 3. *Turn on/off circuit for the V2 and V3 regulators. Energy stored in the inductor turns Q3 off rapidly to reduce storage time and improve efficiency.*

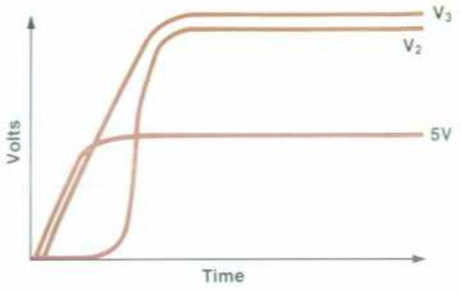


Fig. 4. *Turn-on sequencing is designed for proper operation of semiconductor memory devices.*

The voltage error amplifier for V2 is connected so the V2 output voltage tracks the V3 output. In addition, V2 is inhibited until the V3 output has reached a predetermined level (see Fig. 4). This form of voltage sequencing is required in computer systems for proper operation of semiconductor memory devices. Current limit and overvoltage protection are also provided on V2 and V3.

Other Circuits

In addition to the main output regulator and control circuits, additional circuits are provided for protection and power management. These include:

- Slow turn-on
- Overcurrent/short circuit
- Overvoltage
- Line undervoltage
- Line overvoltage
- Overtemperature
- Inrush current protection
- Inverter peak current limiter
- Internal overload protection
- Remote turn-on/off
- Remote sensing.

Electromagnetic Interference

The power supply is designed to meet the Federal Republic of Germany EMI specification VDE 0871/3.68. A seven-segment EMI filter was designed that incorporates both

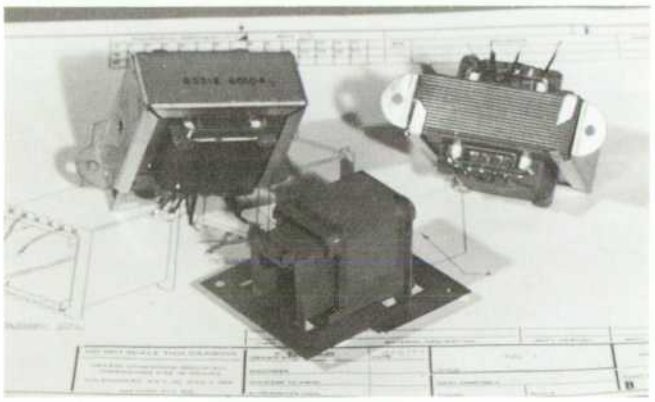


Fig. 5. *Bias transformer uses novel concentric bobbins to achieve required safety spacings.*

SPECIFICATIONS
HP Model 63312F Power Supply

OUTPUT RATINGS: For a total of 550 watts maximum.
- OUTPUT V1: 4.75V to 5.25V at up to 50A
- OUTPUT V2: +11.4V to +15.75V at up to 10A
- OUTPUT V3: −11.4V to −15.75V at up to 10A
- Screwdriver voltage controls and current limit controls are accessible through holes in the panel.
- Output V1 is isolated from V2, V3, and ground and has an independent voltage control.
- Outputs V2 and V3 are complementary tracking outputs adjusted by one voltage control. They have a common return terminal and are isolated from V1 and ground.

TOTAL OUTPUT POWER: 550 watts when ac input is 100-127Vac or 200-250Vac. Derate from 550 W to 475 W as ac input decreases from 100 to 87Vac or 200 to 174Vac.

TEMPERATURE RATING: 0°C to 40°C at full output. Contact your HP field engineer for derating information between 40°C and 70°C. Storage temperature range is −55°C to +85°C.

INPUT RATINGS: 87-127Vac or 174-250Vac, 47-63 Hz, single-phase 1400 VA maximum. Voltage range is changeable on input terminal block.

INPUT PROTECTION: 20A ac line fuse inside the unit.

OUTPUT VOLTAGE EFFECTS FOR V1, V2, AND V3:
- SOURCE EFFECT: 0.1% for input 87-127Vac or 174-250Vac.
- LOAD EFFECT: 0.1% for no-load to full-load current change.
- TEMP. EFFECT: 0.02%/°C, 0°C to 40°C.

PARD (Ripple and Noise)
- OUTPUT V1: 50 mV p-p (20 Hz-20MHz)
- OUTPUTS V2 and V3: 75 mV p-p (20 Hz-20 MHz)

REMOTE SHUTDOWN: Terminals are provided for remote on-off control of all outputs together, and also for outputs V2 and V3 alone. (An open collector low logic level or a contact closure = output off.)

POWER SUPPLY PROTECTIVE FEATURES:
- Output overcurrent protection
- Output overvoltage protection
- Output reverse voltage protection
- Overtemperature protection

PEAK INRUSH CURRENT: Less than 80A peak at turn-on.

COOLING: Integral cooling fan.

REMOTE SENSING: Remote sensing terminals are provided for each output that will maintain nominal voltage at the load by correcting for load lead voltage drop of up to 5%.

DIELECTRIC WITHSTAND TEST VOLTAGE:
- PRIMARY TO ANY OUTPUT: 1500Vrms for 1 minute.
- PRIMARY TO CASE: 1500Vrms for 1 minute.
- ANY OUTPUT TO CASE: 100Vdc for 1 minute.

SAFETY STANDARDS: The power supply meets the requirements of UL478 and is in the Recognized Component Index under Guide QQFU2, File E51529.

EMI CHARACTERISTICS: Conducted interference meets the requirements of VDE 0875/7.71 Level A.

CARRYOVER TIME: At full load and 87Vac input, the output voltages decrease less than 2% during the first 20 ms after input power interruption.

WEIGHT: Net 6.8 kg (15 lb), shipping 8.6 kg (19 lb).

DIMENSIONS: 126 mm H × 207 mm W × 315 mm D (4.96 × 8.14 × 12.4 in).

PRICE IN U.S.A.:
Model 63312F Multiple Output dc Power Supply
List Price, $825.00
OEM 100 unit quantity, $660.00 each.

MANUFACTURING DIVISION: NEW JERSEY DIVISION
Green Pond Road
Rockaway, New Jersey 07866 U.S.A.

Dilip A. Amin

Dilip Amin joined HP's New Jersey Division in 1973, six years after completing his Bachelor of Engineering degree at Maharaja Sayajirao University in Baroda, India, and four years after receiving his MEE degree from Stevens Institute of Technology in Hoboken, New Jersey. Responsible for the circuit design and production introduction of the 63312F Power Supply, Dilip was a project manager for switching power supplies for three years before joining HP. He's named as the inventor on a high-frequency SCR switching regulator patent. Dilip, his wife and two daughters live in Dover, New Jersey. He enjoys hiking, gardening, riding bicycles, sightseeing and discovering new places of interest.

Thane Kriegel

Tim Kriegel came to HP in 1972 with four years experience in power supply development and three years in video systems design. He contributed to the 62605M 500W switching supply as project engineer and served as project leader for the 63312F 550W multi-output switching supply. Tim received his BSEE degree in 1965 from Northeastern University in Boston. A native of the state he still lives in, Tim and his wife and five children, ages 9 to 19, live in Denville, New Jersey. The family enjoys camping together and Tim finds most of his leisure time well filled with the work of renovating an 80-year-old home.

common mode and normal mode filters. The input storage capacitors have low series resistance and inductance to minimize conducted noise to and from the inverter. The inverter transformer has dual primary windings and multi-shielded secondaries to reduce noise conducted through interwinding capacitances. All outputs have Schottky rectifiers and high-frequency filters to minimize output ripple and noise.

Safety

Requirements for high dielectric breakdown strength proved to be an interesting challenge in the design of this computer power supply. Specifications for printed circuit board trace spacings and component spacings made high-density packaging more difficult. In addition, there were internal construction requirements on the safety isolation transformers. New techniques were created in the transformer designs to provide the additional dielectric strength without sacrificing performance and costs. The bias transformer, T5, uses novel concentric bobbins to achieve the required safety spacings (see Fig 5). Inverter transformer T1 and driver transformers T3 and T4 are wound with wire that has additional insulation, and extra taping between windings is provided.

Acknowledgments

The innovative mechanical design features of the 63312F were developed by Don Pauser. Win Seipel designed the novel magnetic components and George McGreen contributed significantly in the area of technical support.

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

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

HEWLETT-PACKARD JOURNAL

JULY 1979 Volume 30 • Number 7

Technical Information from the Laboratories of Hewlett-Packard Company

Hewlett-Packard Company, 1501 Page Mill Road
Palo Alto, California 94304 U.S.A.
Hewlett-Packard Central Mailing Department
Van Heuven Goedhartlaan 121
1180 AM Amstelveen The Netherlands
Yokogawa-Hewlett-Packard Ltd., Suginami-Ku
Tokyo 168 Japan

0200020810&&&BLAC&CA00
MR C A BLACKBURN
JOHN HOPKINS UNIV
APPLIED PHYSICS LAB
JOHNS HOPKINS RD
LAUREL MD 20810

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.