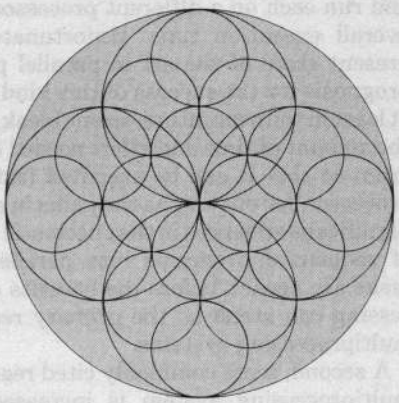
Multiprocessing has distinct advantages—and unique constraints with implications for performance. This survey highlights the architectural strategies of five multiprocessor systems.



SPECIAL FEATURE:

Commercial Multiprocessing Systems*

M. Satyanarayanan
Carnegie-Mellon University

In the decade and a half since the advent of third-generation computers, many multiprocessing systems have been built by both universities and industry. Unfortunately, there seems to be little publicly accessible information on commercially available multiprocessing systems. This survey describes the architectures, system organizations, error recovery facilities, operating systems, and other features of multiprocessing systems made by five major US mainframe manufacturers. Unless dictated by the need for understandability, I have avoided material not directly related to the multiprocessing aspects of these systems. With the exception of the Honeywell 60/66, unclassified quantitative data on performance is unavailable and therefore cannot be given. The concluding section discusses general issues in multiprocessing, drawing upon the information presented in the survey.

Before setting out to study each system in detail, it is appropriate to ask ourselves two questions:

- What is multiprocessing?
- Why is multiprocessing done?

This introduction answers these questions.

What is multiprocessing? The attribute that characterizes a multiprocessing system is the sharing of global memory by the processors making up the system. "Sharing" in this context means that the memory should be addressable by all the processors. Mere linking together of a number of computer systems, as in a computer network, does not result in a multiprocessing system.

To refine our definition, we include two further criteria. First, the processors constituting the multiprocessing system should not be highly specialized. I/O channels, front ends, and other similar specialized processing units often share main memory with a central processor, but such configurations do not fit within our notion of a multiprocessing system. "Highly specialized" is clearly a subjective term. How does one determine whether a processor is highly specialized or not? The view taken here is that a processor is not highly specialized if it can do significant computation on its own, without external assistance.** Independent processing is our second criterion, therefore, and it permits us to classify a system such as the Cyber 170 as a multiprocessing system, even though its central processors and peripheral processors differ radically from each other. The peripheral processors share main memory with the central processors and, in fact, are capable of running the operating system by themselves.

In short, we define a multiprocessor configuration as one which consists of at least two processors satisfying the following conditions:

- the processors share global memory,
- they are not highly specialized, and
- each of the processors can do significant computation individually.

Why use multiprocessing? That multiprocessing is not just an intellectual curiosity but a technique of practical value is clearly demonstrated by the

*This article is taken from a monograph, *Multiprocessors: A Comparative Study*, by M. Satyanarayanan, to be published by Prentice-Hall Inc.

**What is "significant computation" and what is not is once again debatable. In the absence of quantitative criteria, one has to use value judgments at some point. Rather than going through further levels of recursion in this definition, I prefer to appeal to the reader's intuitive notion of "significant computation."

0018-9162/80/0500-0075$00.75

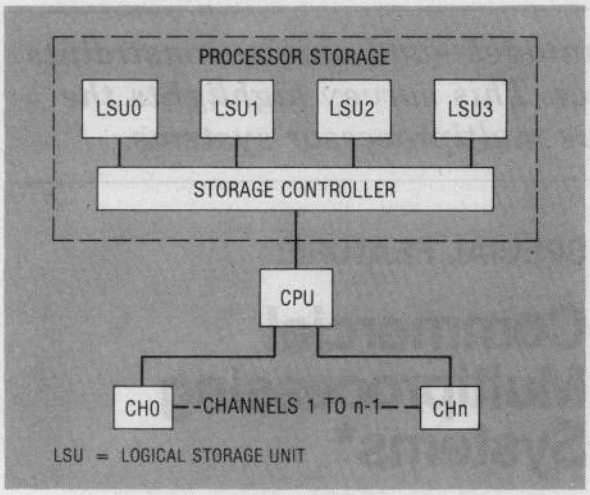


Figure 1. A uniprocessor IBM 370/168 system.

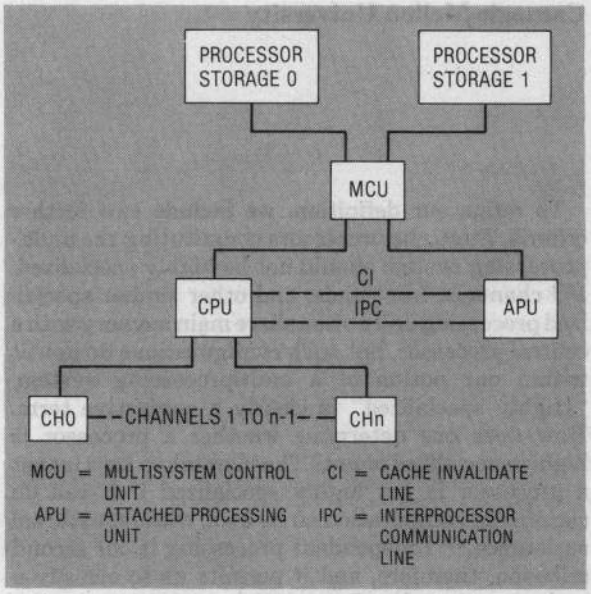


Figure 2. An attached processing IBM 370/168 configuration.

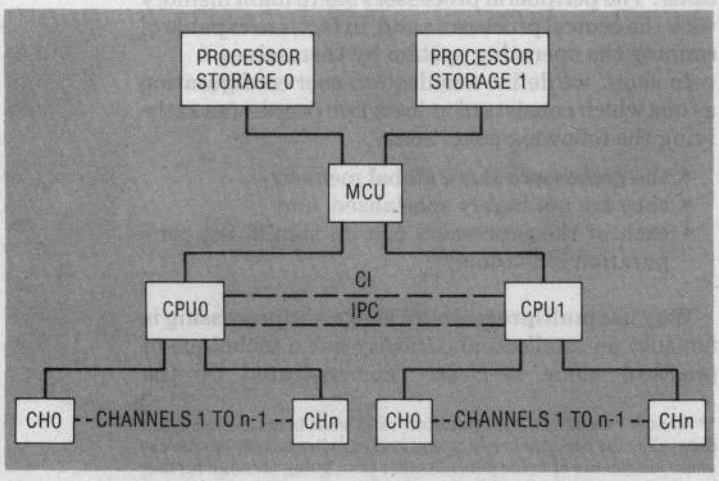


Figure 3. A true multiprocessing IBM 370/168 configuration.

number of commercially available multiprocessing systems. Given such a "proof by existence," it is legitimate to ask the question: "What problems do these multiprocessing systems successfully solve?"

First, given a job which taxes the resources of the largest computers available, it may be possible to split up the job into a number of tasks (or processes) and run each on a different processor, reducing the overall execution time. Unfortunately, given the present state of the art in parallel processing, the prognosis for the success of this kind of application, at least in the near future, seems bleak. In most cases, the amount of time and effort needed to recode a program so that it can be executed faster on a multiprocessor, far outweighs the gains in execution time. Significant advances in the automatic decomposition of sequential programs into parallelly executable tasks are needed before the benefits of parallel processing can stand as the primary reason for using multiprocessing systems.

A second, more commonly cited reason for using a multiprocessing system is increased throughput. Given that the workload at an installation is larger than what the largest uniprocessor can handle, a multiprocessor seems to be a good idea. For example, using two processors as a multiprocessor is usually better than configuring them as uniprocessor systems and partitioning the workload between them. This is true because in a multiprocessor the sharing of hardware resources and processor time tends to smooth out effects due to stochastic variations in job characteristics. To give a trivial example, suppose there are 10 jobs to be run. Splitting them into two groups of five each and running one group on each uniprocessor may, in a pathological case, result in five CPU-bound jobs being run on one processor and five I/O-bound jobs on the other, implying that the former processor will be overloaded and the latter underutilized. Running all 10 jobs on an equivalent multiprocessor configuration, however, will usually result in better performance. This occurs even though each processor in a multiprocessing configuration performs worse than when it is in a uniprocessing configuration—a point I will discuss in the conclusion. Whether or not it is worthwhile to run the processors in a multiprocessing configuration greatly depends on the extent of this degradation as well as on the job characteristics.

Finally, because of the redundancy inherent in a multiprocessor system, it is usually more robust than a uniprocessor system. Failure in any one redundant component (e.g. one processor) is usually noncritical—the system can continue processing with the remaining hardware. The performance in such a case will obviously be degraded. Serviceability is also improved since repairs can be made on defective components while the rest of the system is operating. A major advantage of a multiprocessor over a uniprocessor is that, given the right hardware and software support, a program on a failed processor can be restarted on another processor and continued from a point close to the point at which failure occurred. This implies that the frequency of software checkpoints

can be greatly reduced—no small gain, since checkpointing is notorious for its hogging of computer resources.

In addition to these technical advantages, there are often economic reasons for using multiprocessing systems. Whatever one's favorite reason for multiprocessing is, multiprocessing systems will play an undeniably important role in the computer systems of the future.

The IBM 370 Model 168

The System 370 is a family of computers comprising a wide range of implementations but having a single architectural specification. Case and Padegs[2] describe the S/370 architecture and discuss its evolution. The Model 168, discussed in detail here, is one of the implementations of the S/370 architecture that support multiprocessing. Multiprocessing in the 370/168 comes in two flavors, designated MP, for multiprocessing, and AP, for attached processing. Architecturally these are identical—the same processor instructions are available to handle both situations. From a system organization point of view, however, they differ significantly. For lack of a better term we refer to both of them generically as "MP" and use the term "true MP" when we wish to specify the special subcase.

Overview. Figure 1 shows a uniprocessor 370/168 configuration. The central processing unit contains the instruction decoding and execution units as well as a cache. Main storage is equally divided between four units referred to as logical storage units, and four-way interleaving is present between the LSUs. Peripherals are connected to the system via channels—highly specialized processing units with very simple instruction sets—which operate asynchronously with the CPU.

Figure 2 shows an AP configuration. The structure is very similar to that of a uniprocessor system, with the addition of an attached processing unit. This APU is a processor that is almost identical to the standard 370/168 CPU and differs from the latter only in that channels cannot be attached to it. Since the APU has its own cache, a cache invalidate line is present between the APU and the CPU, to ensure consistency of cache contents. Also present is an interprocessor communication line, used by the APU and CPU to communicate with each other. The term "processor" in the rest of this section will refer to both APUs and CPUs.

A true MP configuration is shown in Figure 3. It can be viewed as being made up of two independent uniprocessor systems, each with its own CPU memory and channels, connected through a multisystem control unit. The MCU handles the routing of memory requests to the appropriate LSU and also contains controls for reconfiguring the system. A cache invalidate interface and interprocessor communication line are present in this configuration as well. Channels are associated with a specific CPU and consequently cannot communicate with another CPU or APU.

The processor. The S/370 has a 32-bit word divided into four eight-bit bytes and a comprehensive instruction set encompassing fixed-point, floating-point, character manipulation, and decimal arithmetic instructions as well as instructions for system control. The architecture is register-oriented, with 16 GPRs—general-purpose registers—and four floating-point registers. Besides these, there are 16 control registers containing control information of relevance to the operating system and the hardware. The state of a processor is defined at any instant by all these registers, together with a 64-bit register called the program status word. The PSW contains status information of interest to user programs—whether interrupts are enabled or not, whether the result of the previous instruction resulted in an overflow or underflow, and so on. Half of the PSW contains the address of the next instruction to be executed and hence plays the role of a program counter.

The S/370 uses a virtual memory scheme which permits programs to address up to 16M bytes regardless of the amount of physical memory present. Translation of virtual addresses to physical addresses is done by the hardware, using page tables set up by operating system software. To speed up the translation process, a cache for page table entries—called the TLB, for translation lookaside buffer—is present. In an MP system, each processor has its own memory-mapping hardware, and the programs running on different processors usually have different virtual address spaces.

Interrupts in the S/370 are grouped into five major classes:

- program interrupts,
- I/O interrupts,
- external interrupts,
- machine check interrupts, and
- SVC—supervisor call—interrupts.

Program interrupts are caused by program-originated conditions such as overflows, invalid addresses, and page faults. I/O interrupts are caused by channels and signify the successful or unsuccessful ending of an I/O operation. Events such as timer interrupts and operator intervention interrupts are classified as external interrupts. One external interrupt that is of special interest in multiprocessing, and which we shall discuss shortly, is an interprocessor interrupt by means of which one processor can cause an interrupt in another processor. An SVC interrupt arises as the result of the execution of a processor instruction, the supervisor call, which is the primary mechanism a program uses to communicate with the operating system.

Regardless of the class, the hardware treats these interrupts in a uniform manner. When an interrupt occurs, the current PSW is stored in a predefined location in main memory (called the old PSW location for that class) and a new PSW is loaded from another predefined location for the new interrupt class. By virtue of this switch in PSWs, the program counter

(which is part of the PSW) is set to a new value—i.e., that of the interrupt handling routine. Information regarding the specific cause of each interrupt is stored by the hardware at predefined locations in memory. The interrupt handling routines use this information to decide on the course of further action.

External, I/O, and machine check interrupts can be disabled by bit settings in the PSW. Interrupts arising when the processor is disabled for the corresponding interrupt class remain pending until the processor becomes enabled for that class. Within these classes, individual interrupt types can be disabled, e.g., individual I/O channels, specific types of external interrupts. A subset of the program interrupts can also be individually disabled. SVC interrupts cannot, however, be disabled.

The interprocessor communication mechanism mentioned earlier is based on a processor instruction called SIGP, for signal processor. Each processor has associated with it a unique, permanent 16-bit address. The value of this address can be obtained by an instruction called SPAD, for store processor address. The SIGP is a three-operand instruction. The first operand specifies a GPR, the second the address of the processor with which communication is desired, and the third an opcode which signifies the nature of the communication. The philosophy behind the SIGP instruction is that a processor P_A executing it should be able to play the role of a (human) operator with respect to the processor P_B with which communication is attempted. Consequently, most of the opcodes serve to perform the functions provided at the system console—e.g., system reset, start, stop, sense. Analogous to information displayed on a console, eight status bits are returned by P_B in the GPR specified by P_A. Most of the order codes are acted upon byP_B without any ensuing interrupt; in fact, a program running on P_B will be totally unaware that a SIGP instruction was executed with its processor as the target. Two of the order codes do cause interrupts on P_B and are handled as external interrupts by P_B. The more commonly used of these is called external call, and is meant to be the primary interprocessor interrupt mechanism. The other, called emergency alert, is generated by the failure recovery component of the operating system only when that component recognizes that recovery is impossible on a processor. An interesting feature of the interprocessor communication mechanism is the presence of a malfunction alert interrupt. A processor automatically generates this interrupt when it encounters a hardware error too severe to execute even error recovery routines. In such a catastrophe, the processor halts and sends a malfunction alert signal to all processors in the system, causing an external interrupt in all of them.

One feature present in the S/370, but not in any of the other systems surveyed here, is a mechanism called "prefixing." The real address range 0-4K is of special significance to the S/370 hardware because the interrupt-related, I/O-related, and error-handling-related storage areas are in this address range. Rather than having such a 4K block held in common for all processors in the system, the S/370 designers chose to have a separate block for each processor in the MP configuration. As a result, real addresses 0-4K have to be mapped to different physical storage locations for each processor. To do this, a register called the prefix register is associated with each processor, and special instructions are provided to read and write it. The contents of a prefix register specify the physical address of a 4K block of memory into which the address 0-4K is to be mapped. Conversely, real addresses within this 4K range are mapped to physical addresses 0-4K. Since each processor has its own prefix register, it follows that by suitably setting each such register, the real address range 0-4K can be made to correspond in each case to a different physical storage block.

Protection in the S/370 is achieved through two independent but complementary mechanisms. Sensitive instructions such as those relating to I/O are classified as privileged instructions. A bit in the PSW determines whether these instructions may be executed. Memory protection is achieved by comparing a four-bit key in the PSW to a four-bit lock settable only by a privileged instruction and associated with each 2K block of memory; if a mismatch occurs, access is denied.

Each processor has three clocks for timing purposes. Two are run independently of their counterparts in other processors. The third clock, called the time-of-day clock, is synchronized so that all time-of-day clocks in an MP environment run in step, thus guaranteeing a unique time stamp for the system.

Channels and I/O. As was mentioned earlier, I/O in the S/370 is performed via channels, which execute commands set up in main memory by the CPU but run asynchronously with the latter. The commands executed by a channel are referred to as channel command words. A number of CCWs may be concatenated to form a channel program; a channel program specifies a sequence of operations to be carried out on one device. To perform I/O on a device, the CPU sets up a channel program in main storage, places a pointer to it in a predefined memory location, and issues a start I/O instruction. The channel then fetches CCWs from the channel program and carries out the desired operations on the device. When the channel program terminates, or when an error is encountered, the channel delivers an I/O interrupt to the CPU and places status information in a predefined memory location. The CPU's I/O interrupt routine handles the termination of this I/O request and initiates the next I/O operation on the device.

Since channels are associated with a specific CPU, it is clear that in an MP situation, one CPU cannot do I/O on channels associated with other CPUs. To eliminate this asymmetry, a mechanism called a two-channel switch can be attached to a peripheral device. This permits the device to be attached to two different channels, possibly on different CPUs. A special CCW permits a data path to be set up between one of the channels and the device; once set up, this path is retained until explicitly released. Thus both

channels can communicate with the device, though only one can do so at a time. In a true MP configuration, two-channel switches can be used to allow both CPUs to have access to shared devices.

I/O interrupts are always handled by the CPU which initiated the corresponding I/O operation. This is true even if the interrupting device is connected via a two-channel switch in an MP configuration—i.e., there is no passing of interrupts between CPUs. Another important point is that only CPUs have channels associated with them; APUs do not. In an AP environment, therefore, all I/O has to be handled by the CPU.

The operating system. Multiprocessing in the S/370 is supported by one of IBM's operating systems called OS/VS2 Multiple Virtual Systems—MVS, for short. Under MVS, each user operates in his own 16M-byte virtual address space independent of all other users in the system. User programs are totally unaware of the number of processors in the system and of the system configuration—i.e., whether it is a uniprocessing, AP, or true MP configuration. There is one copy of the operating system, part of it resident in real memory and the rest paged in on demand. The operating system occupies a part of every user's address space and manages that space completely—one page table per user is maintained for that purpose.

User programs request operating system services by executing an SVC instruction specifying an eight-bit number as an operand. Memory protection is achieved by ensuring that a given page frame is in no more than one user's address space at a time. The hardware lock and key protection mechanism described earlier is used within the operating system to ensure that failure in one component will not contaminate other components.

To perform I/O on a device, a user program sets up a control block containing necessary information and issues an SVC. The interrupt handling routine checks to see if there is a path from the issuing CPU to the device; if so, the CPU queues the I/O request for execution. If a path to the device is not available, the user task is marked as being dispatchable only by the other CPU (a condition called "CPU affinity") and is then suspended. At some future time the other CPU will activate this task and handle its I/O request. On termination of the I/O request, the task is marked as being once again dispatchable on either processor. CPU affinity may also become necessary if a task needs hardware features found only on one processor.

User tasks may create subtasks, and this process may be recursively repeated. Subtasks of a user's job compete for resources with each other, and two subtasks belonging to a user may be running at the same time on two processors, thus reducing the overall execution time of the job.

There are extensive software and hardware error recovery mechanisms specially designed to make use of the inherent redundancy available in MP configurations. When an error is detected on a processor, a machine check interrupt is generated on that processor and the corresponding interrupt handler tries to perform whatever recovery it can. For example, if a memory bank fails, the page tables can be adjusted so that that memory bank is not used any more. If the machine check interrupt handler realizes that recovery is impossible, it issues a SIGP instruction specifying an emergency alert as the order code. The target processor's interrupt handling routine then tries to salvage the job that was running on the failed processor. Sometimes the hardware error is so severe that even the machine check interrupt handler cannot be successfully executed. In that case, the hardware automatically generates a SIGP instruction to all other processors in the system, specifying a malfunction alert as the order code.

The operating system uses a simple linear ordering scheme[8] to avoid deadlock. This scheme works on the assumption that all tasks holding locks proceed at a non-zero speed. If a task holding locks is executing on a processor when the processor fails, a deadlock may exist. The error recovery software is clever enough to incrementally execute all tasks holding locks, in a suitable order, until a sufficient number of locks are released to remove the threat of deadlock.[6]

Certain parts of MVS code must wait for a lock while in a disabled state; i.e., they will have all interrupts masked. As a result, a processor running that piece of MVS code will not respond to emergency or malfunction alert signals, once again leading to a possible deadlock in the case of processor failure. To avoid this, every part of MVS using a disabled wait periodically enables emergency and malfunction alert interrupts for a brief period, guaranteeing that a failing processor's cry for help will not go unheeded.

Reconfiguration of the system is possible, but only with operator intervention. A true MP system can be partitioned into two totally independent uniprocessor systems. In that case, there is no sharing of memory, but part of the memory associated with one CPU can be used by the other CPU. AP configurations cannot be partitioned. Failure of the APU is noncritical, but failure of the CPU results in the failure of the entire system, since APUs cannot do I/O.

The CDC Cyber 170

The Cyber 170 series is a descendant of the CDC 6600, one of the first systems to embody the principle of functional partitioning. By the latter term I mean the splitting up of computing activities into a number of functions and the assigning of different processors to different functions. Such a partitioning usually (but not necessarily) results in significant asymmetry among the processors—the Cyber 170 is no exception to this.

The Cyber 170 series consists of a number of different models, architecturally and structurally similar to each other but differing in performance. The organization of the series' high-end machine, the Cyber 176, differs slightly from that of the rest of the series. The differences, however, do not affect those aspects of the system that are of interest to us, and

hence, for explanatory purposes, we will ignore these differences.

Overview. A Cyber 170 system can best be visualized as two processing subsystems—the central processing subsystem and the peripheral processing subsystem. These subsystems function relatively independently of each other and have access to a common central memory. In addition to the central memory, there is an optional secondary memory called extended core storage, which is a low-speed random-access memory used in conjunction with the central memory to form a two-level memory hierarchy. All these components are connected via a central memory controller, essentially a high-speed cross-bar switch. I/O devices are attached to the peripheral processor subsystem. Figure 4 illustrates these interconnections.

The central processing subsystem. This subsystem consists of one or two central processors, each with its own bus to the central memory controller. There is no interconnection or communication mechanism between the central processors. Each is a powerful processor with a 60-bit word length and operand, index, and address registers. Memory protection is achieved by having two base-limit register pairs in each central processor (one for the central memory and the other for the extended core storage). All addresses generated during instruction execution are checked for validity against the corresponding limit register and, if valid, are relocated by the contents of the base register. There is a program counter, an error-exit register (used to mask or enable program or hardware error exits), and a monitor address register whose function will be described shortly. The instruction set of the central processor is, for the most part, quite conventional and oriented toward numerical computation. There are no I/O instructions and all I/O is handled by the peripheral processor subsystem.

One instruction, the central exchange jump, is worth studying in some detail. The CEJ is essentially a context-switching instruction and its execution results in the current context being saved and replaced by one taken from an area in memory called the exchange package. The EP is a 16-word area located anywhere in memory. Corresponding to each central processor register, and at a predefined offset from the start of the EP, is a save area for that register. When a CEJ is executed, the contents of each register are exchanged with the contents of the corresponding save area in the EP. Thus, by executing a single instruction, all the registers can be saved and loaded with appropriate new values. The location of the EP depends on the mode of the central processor when the CEJ is issued. The central processor executes in one of two modes, depending on the state of an internal flag called the monitor flag. When the MF is set, the central processor executes in monitor mode; otherwise it executes in user mode. The MF automatically gets set or reset by the CEJ instruction and is not under program control. When the central processor is in user mode, the hardware uses the contents of the monitor address register as a pointer to the EP; at the end of the exchange, the MF automatically gets set. In monitor mode, the operand specified as part of the CEJ instruction is used as a pointer to the EP; at the end of the instruction, the MF gets reset. Thus the CEJ is a very fast mechanism for switching back and forth between user and monitor modes.

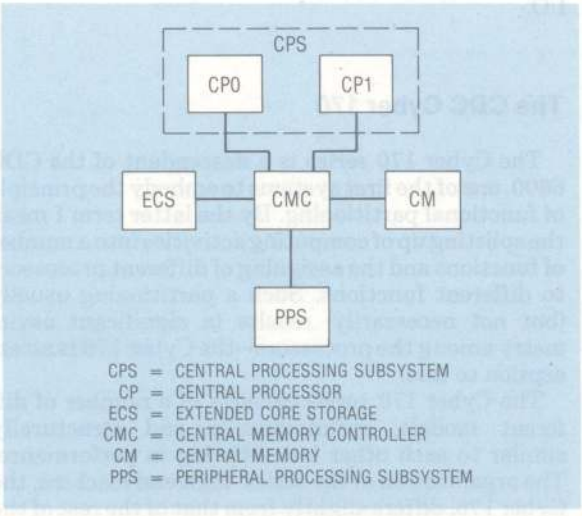


Figure 4. A Cyber 170 configuration with two central processors.

The peripheral processing system. This subsystem comprises 20 peripheral processors, each essentially a minicomputer with an instruction set made up primarily of I/O, logical, and data transfer instructions with a few arithmetic instructions. Each peripheral processor has a local memory of 4K 12-bit words, accessible only to that peripheral processor. There are peripheral processor instructions to transfer data between the central memory and the local memory. Every peripheral processor can use every I/O device in the system, though at any given time one peripheral processor can be in communication with at most one I/O device. There are no interrupts: all I/O is done by the software polling of an I/O device until it indicates that it is ready to receive/transmit data. Data is transferred from an I/O device to a peripheral processor's local memory and then, if necessary, by software to the central memory. There is a 12-bit real-time clock with a resolution of one microsecond. Since there are no timer interrupts, peripheral processors poll the clock to check if a given time interval has elapsed.

The main communication mechanism between the peripheral processors and the central processors is the exchange jump. This instruction comes in two flavors—the EJ, unconditional exchange jump, and the MEJ, monitor exchange jump. Both are similar to the CEJ instruction in that they specify an EP whose contents are exchanged with those of the central processor's registers. In the EJ, the address of the EP is assumed to reside in a peripheral processor register. In the MEJ, the EP address is taken either from a peripheral processor register or from the central processor's monitor address register, depending on the

MEJ's opcode. The MEJ causes an exchange only if the central processor is in user mode; in monitor mode the MEJ is treated as a no-op.

When more than one central processor exists, the one to be interrupted is specified as part of the instruction. If, in a dual-processor configuration, one central processor is executing in monitor mode, a CEJ will remain pending until both processors are in user mode. Under the same circumstances, an MEJ by a peripheral processor to either central processor is treated as a no-op. An unconditional EJ executed at the same time as an MEJ or CEJ can cause both central processors to enter monitor state, and hence can cause them to deadlock. Though a peripheral processor can interrupt either central processor using an EJ, it cannot reset the MF on either. Consequently, once deadlocked, both central processors are doomed to execute in monitor mode unless the system is manually restarted.

The operating system. Three CDC/Cyber operating systems have evolved over the last 15 years—Scope, Kronos, and Nos. The description below pertains to Kronos.

One peripheral processor, designated PP0, is dedicated to the operating system and spends all its time handling supervisor requests from user programs. Another peripheral processor is dedicated to communication with the operator's display console. The remaining peripheral processors form a pool and handle I/O requests. User programs run on the central processors and do not refer directly to the peripheral processors—they are in fact unaware of their presence.

A part of the operating system which does scheduling of central processors and peripheral processors resides in the central memory and is executed by either central processor in monitor mode. Much of the operating system's control information resides in central memory and is used either by a central processor in monitor mode, or by a peripheral processor which transfers it to its local memory before use.

Communication among the various operating system components residing in different central or peripheral processors is done by a message handling system resident in the central memory. Each peripheral processor has an I/O buffer pair permanently assigned to it in central memory. Messages to it are placed in its input buffer and it, in turn, places outward messages in its output buffer. PP0 periodically examines all output buffers to see if any peripheral processors have requested its services.

Since there are no interrupts, all I/O is done via software polling. Direct communication among peripheral processors is done by simulating a peripheral processor-to-I/O device transaction: the responding peripheral processor behaves like an I/O device and transfers data between its local memory and the I/O channel connecting it with the peripheral processor that initiated the conversation.

Performance. The central processor's 60-bit word length and arithmetic instructions make the Cyber 170 eminently suited for scientific computation, where numerical accuracy is vital. The fast context-switching capability makes it good for situations where frequent task switching occurs.

Quantitative performance measures are hard to come by in the literature. One source[9] states that the performance of the Cyber 170 series varies from 0.8 MIPs for the Cyber 171 to 15 MIPs for the Cyber 176. Symmetric dual-processor configurations are stated to have about 1.5 times the MIPs of corresponding single-processor systems.

The Honeywell Series 60 Level 66

The Honeywell Series 60 does not represent one computer system or a family of systems. Rather, it represents a set of families, each referred to as a level, with members of a family referred to as models of that family. There are broad architectural differences among the various levels; however, models of a given level are architecturally similar and differ only in implementation and performance. The level chosen here is Level 66, close to the top of the Series 60 line.

For convenience, we shall refer to the Honeywell Series 60 Level 66 system as the 60/66 throughout the rest of this section.

Overview. The basic structure of a uniprocessor 60/66 system consists of a central processor, an I/O multiplexer with peripherals attached, and a system controller connecting a pair of memory modules to the rest of the system (Figure 5). The system controller acts as a coordinator and plays a crucial role in the routing of memory accesses, interrupts, and other communications among the various system components.

A multiprocessor 60/66 can be visualized as a number of uniprocessor systems connected to each other via their system controllers (Figure 6). As the figure indicates, the connections form a network in which every central processor and every I/O multiplexer is connected to every system controller. In such a system, one central processor is nominated as pro-

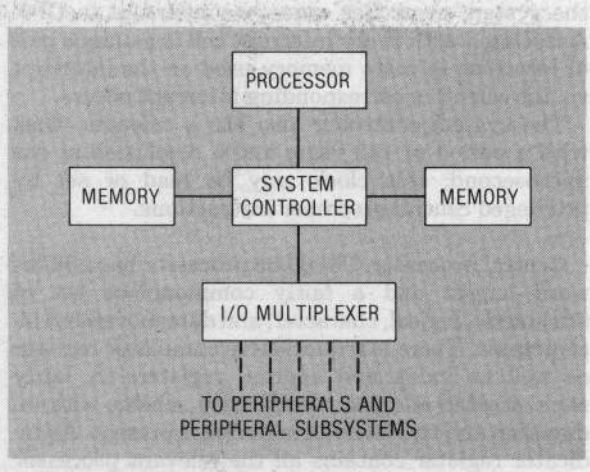


Figure 5. A uniprocessor Honeywell 60/66 system.

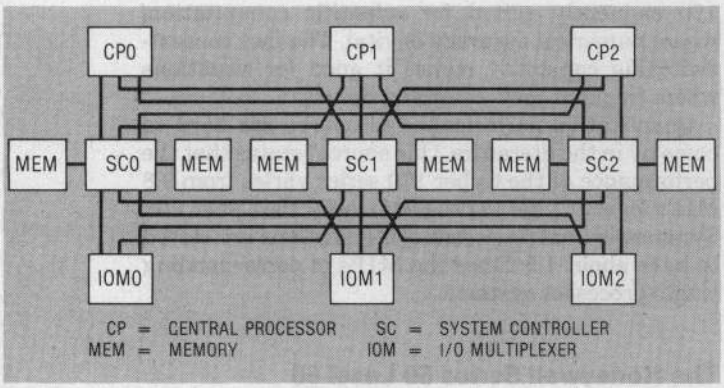


Figure 6. A multiprocessor Honeywell 60/66 system.

cessor 0—CP0—and is distinguished from the others in that all external interrupts are routed to it by the hardware; this solves the problem of each peripheral having to decide which central processor it should interrupt. Up to a maximum of four central processors (and the corresponding number of I/O multiplexers and system controllers) can be included in a multiprocessor system.

Since it plays such a pivotal role in the system organization, we should begin our examination with the system controller.

The system controller. This component has two main functions: it acts as a storage controller for a specific pair of memory modules, and it acts as an intelligent switch for the various components connected to it.

Each system controller has eight ports to which are connected two memory modules, three central processors, and three I/O multiplexers, one to a port. Every system element attached to a port can access the memory associated with that port's system controller. When more than one element tries to access the same memory module, the corresponding system controller resolves the conflict.

Each system controller has 32 interrupt cells associated with it, each of which may be set by an element attached to a port. When an interrupt cell is set, the system controller causes an interrupt in CP0. Associated with every interrupt cell is a unique pair of locations in main memory used as the interrupt vector when the corresponding interrupt occurs.

The system controller also has a calendar clock with a period of 142 years and a resolution of one microsecond; this clock may be read or set by privileged central processor instructions.

Central processor. The 60/66 processor has a 36-bit word length and a fairly commonplace set of arithmetic, logical, character, and data movement instructions. There is a quotient-accumulator register as well as index and address registers. A fairly sophisticated address modification scheme with indirection and two levels of indexing is present. An indicator register contains all the relevant processor status bits such as condition codes from the previous instruction and what program events are enabled.

The processor can be in either master or slave mode. In master mode, all instructions may be executed, whereas in slave mode an attempt to execute certain instructions results in an illegal procedure fault (faults and their processing are described below). Address relocation procedures also differ in the two modes.

Events that disrupt the normal sequence of instruction execution are classified in two categories—interrupts and faults. An interrupt is a request by a system controller for service. A fault is a condition, usually arising in the processor itself, which requires special handling. A signal from an I/O device on completion of a previously requested I/O operation is an example of an interrupt, while overflow in an arithmetic instruction is an instance of a processor-originated fault requiring special processing. Interrupts and faults are handled in an identical manner by the central processor. Corresponding to every type of interrupt or fault is a unique pair of locations in main memory called the interrupt vector. When an interrupt or fault occurs, the central processor enters master mode and executes the instructions stored in the corresponding interrupt vector. The program counter is not altered unless an explicit transfer instruction is one of the instructions in the interrupt vector. At the end of the interrupt sequence, execution resumes at the location pointed to by the program counter. The central processor mode remains what it was before the interrupt or fault occurred, unless it was explicitly altered in the interrupt sequence or a transfer was present in the interrupt vector, in which case the central processor remains in master mode. The above procedure is thus equivalent to a hardware-initiated two-instruction subroutine call in which one can examine an interrupt or fault and decide whether further processing is necessary.

Protection and isolation in the 60/66 are achieved by a base/limit register mechanism. Though the 60/66 can have up to 1M words of physical memory, its 18-bit addressing structure allows only 256K of that memory to be addressable at a time. The specific 256K "window" that is addressable is determined by a base register whose contents are added to every address generated by the processor. There are, in fact, three base address registers—one for use in slave mode and the other two for master mode, thus permitting a program in master mode to access two different 256K windows.

In a multiprocessor configuration there is only one contiguous, real address space for the entire system. Each central processor has its own set of base address registers. Thus programs executing on different central processors can refer to different extents of the real address space, or, if desired, share the same (or overlapping) parts of it by appropriately setting the respective base registers. Each system controller has a different set of interrupt vectors corresponding to it, thus ensuring that each interrupt cell in a multiprocessor system is indeed associated with a unique interrupt vector.

I/O operations are done using the CIOC—connect I/O channel—instruction. A channel is a connection between an I/O multiplexer and a peripheral device or subsystem. Corresponding to every channel in the system is a unique location in memory called a channel mailbox. To initiate I/O on a channel, a central processor first fills this mailbox with pertinent information for the channel and then issues a CIOC for that channel. The channel reads its mailbox, performs the operations requested, and presents an interrupt (via the I/O multiplex and system controller to which it is connected) to CP0.

Communication between processors is possible in a multiprocessor system since processors themselves can be attached to channels. Consider two central processors, P_1 and P_2, connected to channels C_1 and C_2. If P_1 wishes to pass a message to P_2, it can place a message in C_2's mailbox and issue a CIOC for C_2. The hardware recognizes this as an interprocessor communication and causes a special fault (connect fault) to occur in P_2. P_2 now executes the corresponding fault service routine and reads the message sent to it. It can send a reply by a method symmetric to the above.

Each central processor has an interval timer which can be set in master mode and which causes a timer fault when the specified time has elapsed. The fault occurs only in slave mode; if the central processor is in master mode, the fault remains pending until it enters slave mode.

I/O multiplexer. This component is connected, through I/O channels,* to every system controller and to a variable number of peripherals or peripheral subsystems. Each memory access required by a channel is performed via the I/O multiplexer, which routes it to the appropriate memory module and performs bounds checking on it.

The I/O multiplexer permits scatter-gather of data—i.e., the ability to read/write a block of data from/to noncontiguous locations in memory. An I/O multiplexer can set an interrupt cell in a system controller and hence cause an I/O interrupt in CP0. In short, the I/O multiplexer acts as an intelligent interface between the I/O channels and the system controllers.

Operating system. The 60/66's operating system is called GCOS—General Comprehensive Operating System. GCOS is a multiprogramming operating system and, on the 60/66, supports multiprocessing. Under GCOS each job executes in its own address space; the relocation of a job's address space to its image in real memory is done by the base address register, as described before. GCOS gives a job a uniprocessor view of the system, regardless of whether the system is really a uniprocessor or multiprocessor—i.e., a job will be unaware of the number of central processors in the system on which it is running. Scheduling is done on a time-sliced basis using each central processor's interval timer.

User programs execute in slave mode. Requests for operating system services are made by executing the MME—master mode entry—instruction; MME is very similar to the SVC instruction in the S/370 and causes a program-generated fault to appear on the central processor which executed the instruction. The operating system's MME fault handling routine routes the request to the appropriate service routine, which, on completion, returns control to the user program in slave mode.

Resource management and scheduling are done on a system-wide basis. There is one copy of the operating system code resident in memory and relevant parts of it are executed by any of the central processors when needed. The asymmetry in hardware (i.e., the fact that one central processor is the control processor) is reflected in the software only in I/O handling. I/O operations may be initiated by any central processor. For each channel there exists a global queue of I/O requests awaiting attention on that channel. When a channel becomes free, the control processor schedules a request from the queue and issues a CIOC. When the channel completes the request and interrupts CP0, the latter fields the interrupt and marks the request as satisfied; tasks waiting for this event are enabled and will be scheduled in the usual manner the next time the dispatcher is activated. The fact that only CP0 can handle interrupts does not significantly complicate the operating system code. There is, after all, only one copy of the code, and asymmetry in hardware will be reflected only in the fact that certain parts of the code are always executed by only one central processor (i.e., CP0) and not by the others. The routing of interrupts to CP0 applies only to (external) interrupts; faults are handled on the processors on which they are generated.

Mutual exclusion among critical sections of operating system code and data structures is obtained through locks. Certain central processor instructions are designed for this purpose: examples are the LDAC, load accumulator and clear memory location; the LQAC, load quotient and clear memory location; and the SZNC, set zero and negative indicators and clear location. All of these access and then clear a memory location in one, indivisible sequence.

Communication among various tasks is accomplished by an "intercom" system, a component of GCOS that does virtual I/O between two tasks. A task T_1 wishing to send a message to a task T_2 does a write operation via intercom. When T_2 issues a read to intercom it receives the message sent by T_1.

On-line reconfiguration of a 60/66 system is possible through operator commands. Central processors (with their system controllers and I/O multiplexers) can be added to or deleted from the system, and a multiprocessor system can be partitioned into two or more uniprocessor or multiprocessor systems. All such reconfigurations require operator intervention; the system cannot automatically reconfigure itself in the event of a failing element. Such automatic recon-

*These I/O channels are essentially direct memory access controllers.

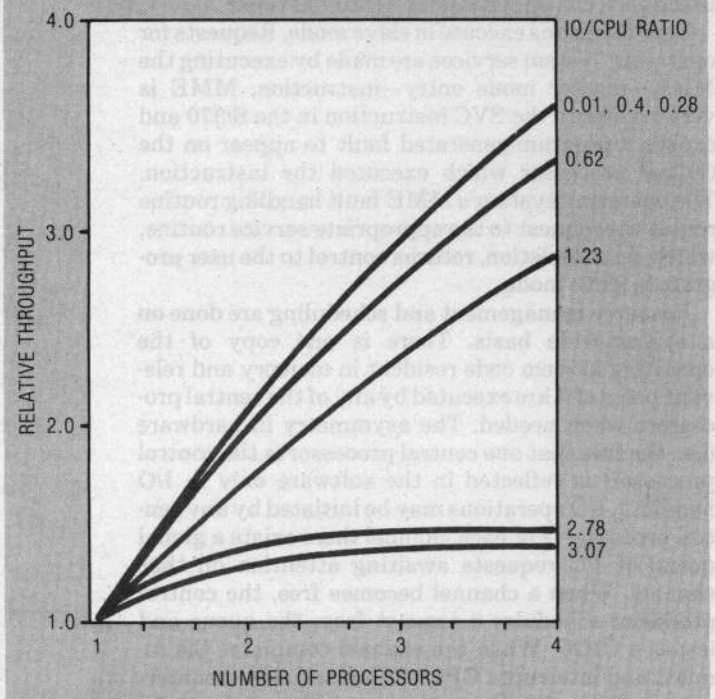


Figure 7. Honeywell 6060 performance (interleaving on).

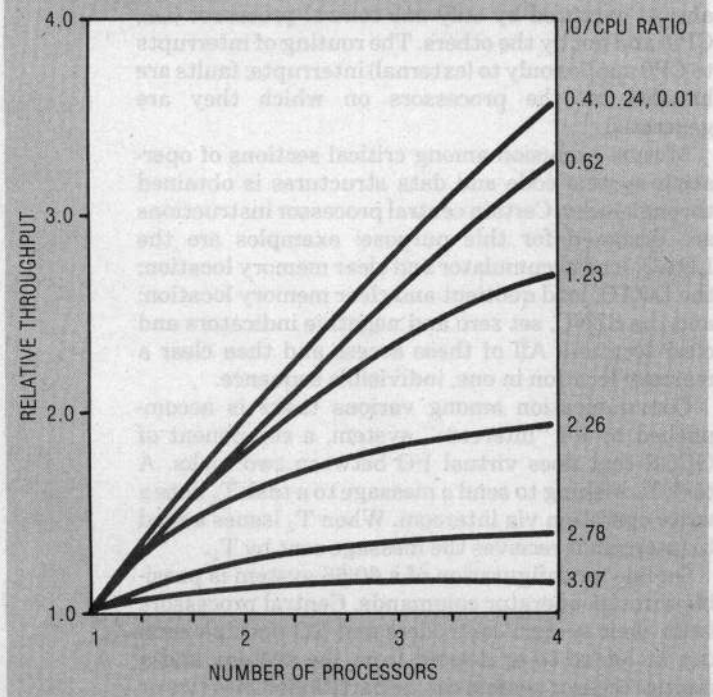


Figure 8. Honeywell 6060 performance (interleaving off).

figuration would be especially difficult in this system, since the control processor has to be manually designated and switches set accordingly.

Performance. A reasonably exhaustive search of the literature failed to turn up any information on the performance of the multiprocessor 60/66 systems. However, a study for the Series 60's predecessor, the Series 6000, does exist. The organization of the 6060 is similar to that of the 60/66. Although the *absolute* measures of performance are likely to be different, this similarity makes it possible to show the *relative* improvement one might expect in going from a uniprocessor to a multiprocessor system. The study[22] measured throughput as a function of the number of processors in the system, with and without memory interleaving (Figures 7 and 8). The evaluation was done using a synthetic job mix in which the I/O-to-CPU time proportion for a job could be varied. The curves clearly indicate the ineffectiveness of adding more processors in I/O-bound situations, but the value of such additions in CPU-bound cases. The same study quotes the following (previously unpublished) Honeywell figures for relative throughputs of 6060 multiprocessor systems:

No. of CPUs	*Relative throughput*
1	1.0
2	1.8
3	2.4

The Univac 1100 Model 80

The Univac 1100 series is a descendant of the venerable 1106, 1107, and 1108 systems and, consequently, retains most of their architectural features. The survey by Borgerson et al.[23] gives an excellent description of Univac system evolution from the 1107 to the present-day 1100 series. This series comprises many models, architecturally identical but differing in performance. The Model 80, discussed here, supports up to four central processors.

Overview. Figure 9 shows a maximally configured, single-processor 1100/80 system. The system is organized along fairly conventional lines, with a central processor and an I/O unit attached to main storage units through a storage interface unit. The storage interface unit contains a cache to speed up memory references. Peripherals are connected to the storage interface unit through channels (similar to IBM channels) housed in the I/O units.

The configuration shown in Figure 9 is called a 1×1 system, since there is one processor and one I/O unit. In general, 1100/80 systems are designated as $M \times N$ configurations, where M is the number of processors and N the number of I/O units. Configurations in the range 1×1 to 4×4 are possible, though not all combinations of M and N are permitted.

In a 2×2 system (Figure 10), two processors and two I/O units are connected to a storage interface

unit. There is still only one cache, common to both processors and located in the storage interface unit.

Figure 11 depicts a 4×4 system. A second storage interface unit, with its own independent cache, is now present and connected to the two additional processors and I/O units. The two storage interface units have a cache invalidate interface which ensures that if both caches contain copies of the same data, altering the copy in one cache will cause the corresponding copy in the other to be marked as invalid.

As the figures show, main memory is a common resource for all processors and I/O units and is accessed by them via the corresponding storage interface units. There can be up to four main storage units, each containing from 512K to 1M words of memory. Each main storage unit is connected to both storage interface units (if two exist) and can be two-way interleaved.

Processors are connected to each other by interprocessor interrupt interfaces, which permit a processor to cause an interrupt in any other processor. An I/O unit is electrically connected to only one storage interface unit and to the processors on that storage interface unit. As a result, a processor can handle I/O only on I/O units connected to the same storage interface unit as itself.

In addition to the components mentioned above is a stand-alone unit (not shown in the figures) called the system transition unit, which contains controls necessary for system reconfiguration.

Central processor. The 1100/80 processor has a 36-bit word length and a reasonably rich repertoire of fixed-point, floating-point, data movement, and character manipulation instructions. The architecture is essentially register-oriented, with separate index registers and accumulators. Most double-operand instructions have one operand in a register and one in memory.

Central to the architecture of this system is a set of 128 36-bit words called the GRS—general register set. Programs can address 16 index registers and 16 accumulators. There are two sets of each—one for use by user programs (the user set) and the other for use by the operating system (the EXEC set). There are also two sets of miscellaneous registers used for specifying masks, repeat counts, etc. Besides these registers, the GRS contains a number of elements used only by the operating system: a real-time clock, pointers to descriptor tables (described later), and areas used by the hardware to save information during interrupts.

Included in the processor, but not as a part of the GRS, is a register called the DR—designator register. The DR is a collection of miscellaneous bits, some of interest to user programs, others used only by the operating system. In addition to condition code bits (indicating the results of the previous instruction), the DR contains a number of control bits indicating certain program conditions such as overflow and divide check, enabling or disabling interrupts, specifying whether user or EXEC registers are to be used, and so on. The DR may be loaded by a processor in-

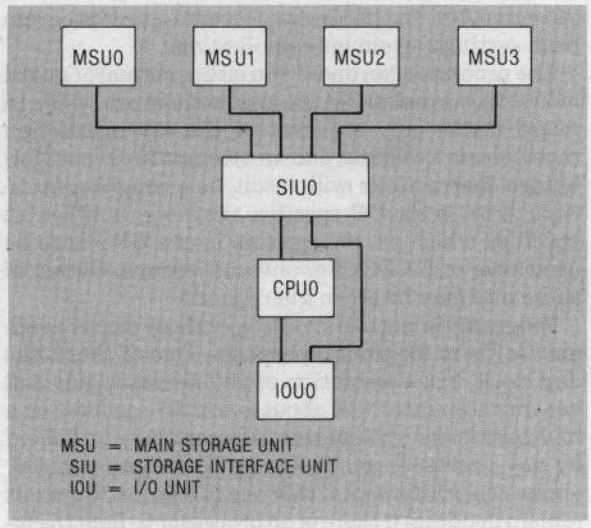


Figure 9. A 1×1 Univac 1100/80 configuration.

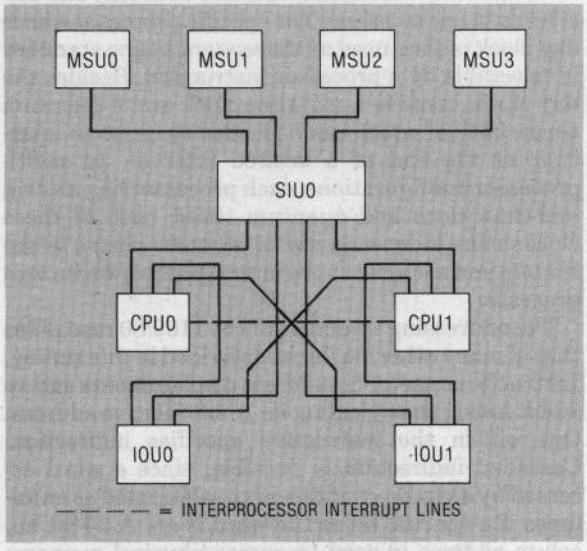


Figure 10. A 2×2 Univac 1100/80 configuration.

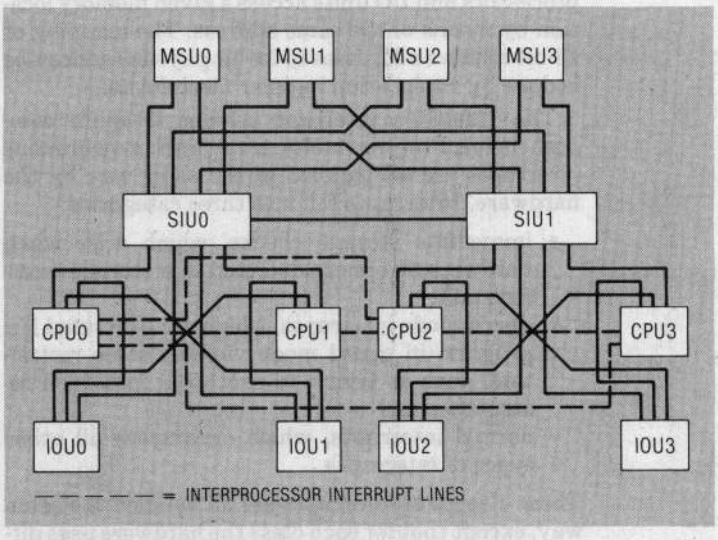


Figure 11. A 4×4 Univac 1100/80 configuration.

struction, but the hardware prevents a user program from setting certain bit combinations.

The processor can operate in either normal or guard mode. In normal mode, all instructions are valid. In guard mode, only a subset of the full instruction repertoire is available, and an attempt to execute forbidden instructions will result in a program interrupt. A bit in the DR specifies the mode; another bit specifies which set of registers in the GRS is to be used (user or EXEC). For obvious reasons, neither of these bits may be set in guard mode.

Programs in normal mode have three clocks available to them for timing purposes. One of these, the day clock, has a resolution of 200 microseconds and generates an interrupt about every 6.5 seconds. In a multiprocessor system this interrupt may be fielded by any processor which has its interrupts enabled. Unike other interrupts, this one is lost (i.e., does not remain pending) if there is no processor to field the interrupt. Further, in a multiprocessor system a hardware interlock mechanism ensures that only one day clock is in use at a time. The specific processor whose day clock is thus used as the system's time standard is selectable by a processor instruction. Besides the day clock, there is a real-time clock and a quantum timer, both of which can be loaded to cause an interrupt at the end of a desired interval. In multiprocessor configurations, each processor has its own real-time clock and quantum timer; each of these clocks runs independently of all other clocks in the system and each can cause interrupts only on its own processor.

The addressing structure of the 1100/80 resembles that of many other machines described in this survey. Instructions use 16-bit address displacements and an 18-bit index, thus yielding an 18-bit relative address. One bit in the instruction specifies indirection. Cascaded indirection is possible, since a word accessed by an indirect address is itself treated as an address if a specific bit in the word is on. A 24-bit absolute address is used to access physical memory; these addresses are system-wide in the sense that all processors and I/O units access a given memory location by means of the same address. The mapping of 18-bit relative addresses to 24-bit physical addresses is done by a relocation register mechanism.

The 1100/80's interrupt scheme is quite comprehensive. Program faults, traps, and asynchronous interrupts are all handled in the same way by the hardware. Interrupts fall into three categories:

- immediate storage checks, which arise when hardware errors are detected in a storage interface unit;
- guard mode interrupts, which are generated if a program in guard mode violates some restriction, such as trying to execute a forbidden instruction; and
- normal interrupts, which encompass all other types of interrupts.

These classes of interrupts are all handled the same way, except that for each class the hardware uses different locations in the GRS as save areas. A set of 56 contiguous storage locations is used in a special way by the hardware; most are used as interrupt vectors, one such vector being assigned to each type of interrupt. The location of this special area of memory can be specified by a processor instruction and by switch settings on the system transition unit. In a multiprocessor system, there is only one such set for the entire system, thus implying that all interrupt routines must be reentrant or have locks ensuring mutual exclusion. When an interrupt occurs, the DR, the current value of the program counter, and the status word giving details specific to each type of interrupt are stored by the hardware in predefined locations in the GRS. All DR bits are set to zero, except those specifying relocation suppression and EXEC register use in the GRS. An unconditional branch to the interrupt vector corresponding to this interrupt is then made. The (single) instruction at this location is usually a branch to the appropriate interrupt service routine. All further interrupts are disabled until explicitly enabled. The interrupt service routine uses the interrupt status (saved in the GRS) to obtain information needed to process the interrupt. Control is returned to the interrupted program by a single instruction, which loads the DR and branches to the point of interruption.

Communication between processors is achieved by an interprocessor interrupt scheme. A processor number from 0 to 3 is associated with each processor to uniquely identify it. This number is defined in hardware and may be obtained by software by executing a store processor ID—SPID—instruction. A processor P_A wishing to interrupt processor P_B can issue an initiate interprocessor interrupt instruction specifying B as the operand. An interrupt on processor P_B is then generated and, when enabled, causes P_B to execute the corresponding service routine. P_B can determine the identity of the processor causing the interrupt from the interrupt status word saved in the GRS. If a processor has more than one interprocessor interrupt outstanding at any time, the order in which such interrupts are serviced is determined by a hardware-defined static priority scheme.

Storage interface unit. As the name implies, this component acts as an interface between storage units on the one hand and processors and I/O units on the other. Each storage interface unit can connect to up to two processors and two I/O units. Hence, in a fully configured (4×4) system two storage interface units are present (Figure 11). Each storage interface unit contains its own four-way, set-associative, store-though cache, whose size can vary from 4K words to 16K words. As mentioned before, a cache invalidate interface exists between storage interface units to ensure consistency. In contrast to most other systems, the I/O unit in the 1100/80 is connected to cache rather than directly to main memory. Moreover, cache is associated with a storage interface unit rather than with a processor, resulting in cache being shared by two processors.

Input/output. I/O in the 1100/80 is done through asynchronous channels, similar to those in the IBM S/370. As in the S/370, there are byte-multiplexer and block-multiplexer channels.*

An I/O unit has a control section, which interfaces with a storage interface unit, and a channel section, where up to eight channels may exist. The channels themselves are connected to peripheral devices and peripheral subsystems via bidirectional links.

For each processor in the system, there are three words reserved in main storage for use in I/O: a channel address word—CAW, an interrupt address word—IAW, and a channel status word—CSW. To start an I/O operation, a processor sets up a channel program in main storage and writes into its CAW a word containing an opcode, channel address, and device address. This initiates the specified operation. On completion of the I/O request, a channel makes an interrupt request via its I/O unit. Either processor connected to the same storage interface unit as this I/O unit may respond, with hardware interlocks guaranteeing that no more than one processor responds. The I/O unit now writes into the responding processor's CSW and IAW. The IAW identifies the interrupting channel and device, and the CSW contains status information for the interrupt. In case both processors have disabled interrupts, the interrupt remains pending until a processor accepts it.

The system transition unit and reconfiguration. The system transition unit is a stand-alone unit containing all controls pertaining to system configuration and partitioning. An 1100/80 installation with more than one processor and I/O unit can be split into two totally independent computer systems (called "applications" in Univac terminology). If only one storage interface unit exists, as in a 2×2 system, each application uses one half of that storage interface unit, with the cache equally divided between the halves. The reconfiguration of a one-application system into a two-application system, or vice versa, has to be done manually. Software running on an application can obtain information about the configuration of that application by issuing processor instructions, which elicit this information from the system transition unit. Attaching a peripheral to one application or the other is done by manual switch setting, though in some cases it can be done by software commands.

The system transition unit also plays a crucial role in error recovery. Two separate initial load paths, specifying the processor, I/O unit, channel, and device to be used in the initial loading of the system, may be set up on the system transition unit; either of them can be chosen at load time. In addition, there is an auto-recovery option that can be used to automatically recover from system software crashes and transient hardware problems. When this option is enabled, a special timer called the ART—auto recovery timer—counts down continuously from a preset initial value. The system software must periodically reset the ART before it goes to zero. If the ART does go to zero, a system error is assumed to exist and a reloading of the system is attempted using one of the load paths. If this fails, reloading is attempted using the other load path. If two such attempts on each load path fail, a permanent error is assumed to exist and the system halts.

*A third type of channel—called the word channel—exists mainly for compatibility with earlier Univac systems. We shall not discuss word channels here.

The operating system. EXEC, the 1100/80's operating system, is a typical example of an operating system which was developed for a third-generation computer in the mid-60's and which has evolved over the years. It supports multiprogramming and multiprocessing. User programs run in guard mode and, unless explicit sharing is requested, user memory areas are mutually exclusive. The specific configuration on which a user program runs (i.e., the number of processors, the number of I/O units, etc.) is transparent to the program. User requests to EXEC are made by executing an "executive request interrupt" instruction. This causes an interrupt on the processor on which the instruction was executed; the operating system's interrupt service routine handles the request and returns to the user.

To request an I/O operation, a user program executes an executive request interrupt with suitable operands. If the request is to an I/O unit on the same storage interface unit as the processor running the user program, that processor initiates I/O on the appropriate channel and device. Otherwise, it signals a processor on the other storage interface unit (by the interprocessor interrupt mechanism) to initiate the I/O operation. As described earlier, the interrupt, on completion of this I/O request, is handled by one of the processors on the same storage interface unit as the interrupting I/O unit.

Only one copy of EXEC resides in main store. Most parts of the code are reentrant; the few that are not use locks to ensure mutual exclusion. Indivisible instructions, such as test and set, test and clear, etc., are available for implementing such locks. Scheduling of user tasks is done using a single ready list for all processors. Error recovery is handled by the operating system, using the hardware auto-recovery feature described above.

The Burroughs B7700

The announcement of the Burroughs B5000 in 1961 was a major landmark in the evolution of computer architecture, for it signified the first major system exclusively designed to support a high-level language (in this case, Algol-60). The B7700 displays many of the distinctive features of the B5000. Organick[31] presents a lucid, though somewhat abstract, explanation of how Algol and Algol-like programs are run on the Burroughs machines; I recommend it to those unfamiliar with stack-oriented architectures.

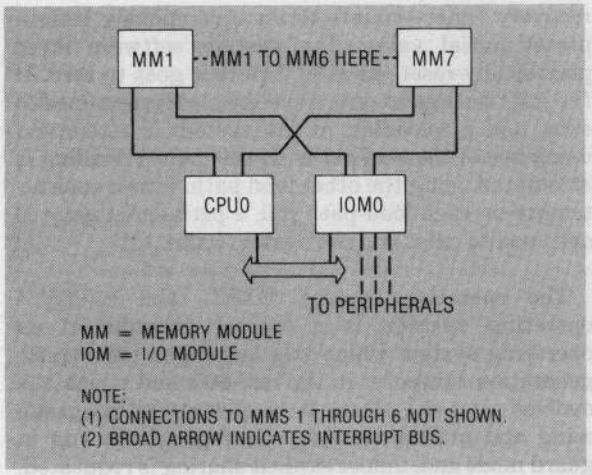


Figure 12. B7700 configuration with minimum number of requesters.

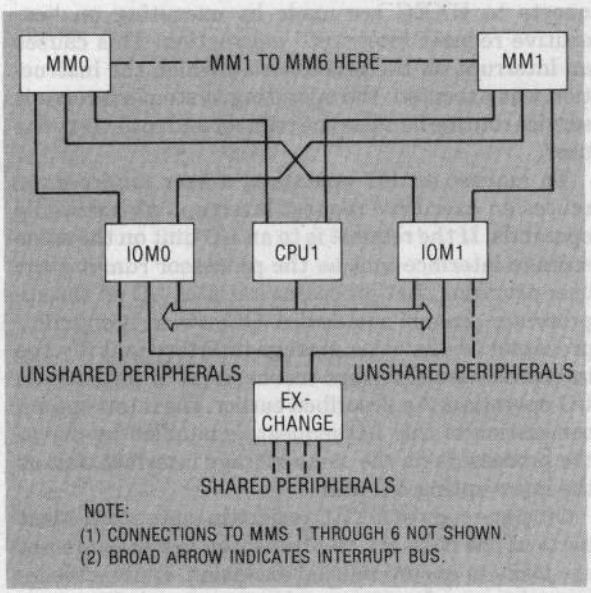


Figure 13. B7700 configuration with one CPU and two I/O modules.

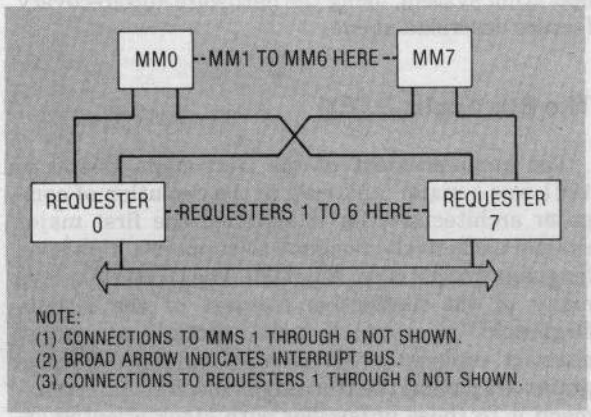


Figure 14. Maximally configured B7700 system.

Overview. A simple B7700 system comprises three major units—the central processor, the input-output module, and the memory module (Figure 12). There can be one to eight memory modules, each containing its own control unit and either 128K words or 256K words of memory (up to a system maximum of 1M words). Peripheral devices are connected to the system via an I/O module.

The system in Figure 13 has one CPU but two I/O modules. Each I/O module and CPU is connected to every memory module by a separate memory bus. There is a single interrupt bus, to which are connected both I/O modules and the CPU. Since both CPUs and I/O modules can access memory, Burroughs refers to them collectively as "requesters of memory." Note, in Figure 13, the presence of an exchange, essentially a circuit switching unit which sets up data and control paths between any of the I/O modules and any of the devices connected to the exchange. The exchange thus permits sharing of devices across I/O modules. Obviously, at any given instant a peripheral can be operated only under the control of one I/O module.

Figure 14 shows a maximally configured B7700 system. Up to eight requesters of memory (with at least one CPU and one I/O module) may be present, each connected to every memory module. However, there is still only one interrupt bus. This bus is symmetric; i.e., any requester can interrupt any other requester.

Unlike some of the other multiprocessor systems examined here, there is no central facility for routing memory requests to the appropriate memory module. Such routing is done by each requester, with conflicts resolved at the memory modules.

Central processor. The B7700 is based on a 48-bit word size, subdivided into six eight-bit bytes. Opcodes* are of variable length (from one to 12 bytes), with each byte referred to as a program "syllable." Every word in main memory is associated with a three-bit tag identifying the generic type of the contents of that word. These tag bits are *not* present just for use by software, but are interpreted and implicitly used by the hardware during program execution.

Program execution in the B7700 uses stacks extensively. At any instant there is a unique stack associated with the job being executed on a processor. This stack consists of three sections,** physically disjoint in memory but forming a single logical entity (Figure 15). The lowest of these sections, common to all jobs in the system, is called the MCP stack since its contents are used by the operating system. (MCP, for Master Control Program, is the name Burroughs applies to its various operating systems.) The second section contains a descriptor (called a segment descriptor) for every segment that will ever be used by the currently executing program and is therefore called the "descrip-

*"Operators" in Burroughs terminology.

**This statement and the corresponding discussion are strictly valid only as long as the job does not do multitasking. If multitasking is done, the stack structure will resemble a "cactus stack."[30]

tor segment." A segment descriptor is a word containing the size and the address, in main or auxiliary storage, of the segment it refers to, and information as to whether this segment is in main memory. References to this segment are routed through the descriptor. The hardware automatically alerts MCP if the segment is not in main memory when an access is attempted; MCP then brings the segment into main memory and resumes the original program. This mechanism is similar to a paging scheme, except that segments are of variable length. If two or more jobs are executing the same program, they share the same descriptor segment, thus ensuring the presence of only one copy of the program code. This description of descriptor segments and program sharing is necessarily superficial; Organick[31] presents a good, detailed discussion of these aspects. The third stack section is unique to each job and consists of the activation records of all procedures and blocks which the job has entered but not yet exited. The first word of each activation record is a special word called an MSCW—mark stack control word—and is identified by a special tag value. This stack section grows and shrinks dynamically during program execution as procedures are entered and exited.

During program execution, most memory references are made to locations at or close to the stack top. To take advantage of this locality of references, the CPU contains a set of 32 locations, called the stack buffer, which are in one-to-one correspondence with the top 32 stack locations in memory. This stack buffer is totally transparent to software, and its management is completely handled by hardware—all that the software "notices" is that stack operations are much faster than if the stack top was in main memory. Besides the stack buffer, the CPU has a 16-word associative memory buffer. This buffer holds copies of data which are not at the stack top, but which have been frequently accessed. As in the case of the stack buffer, the management of the associative buffer is done wholly by hardware; the software is unaware of the buffer's presence. The stack buffer and the associative buffer thus constitute what would be called a cache in more conventional architectures.

Elements on the stack are addressed using two-dimensional addresses of the form <*lexicographic level of element, offset of element within its activation record*>. A set of 32 registers called display registers resolves these two-dimensional addresses. These registers are set up so that the i^{th} register contains the starting address of the activation record at lexicographic level i. Translation of two-dimensional addresses into physical memory addresses is done by the hardware, using the display registers.

In the B7700, program code never contains explicit references to absolute memory addresses. The CPU contains two registers called top-of-stack locations. These locations are loaded by popping the stack, and stored by pushing onto the stack. Many operators implicitly assume that their operands are in the top-of-stack locations. References to stack items below the stack top are made via IRWs—indirect reference words—previously pushed onto the stack. An IRW is essentially a relative pointer, giving the display register number and offset of its target. An IRW can point to another IRW, the latter to yet another, and so on. When an access is made using an IRW, the hardware follows the chain of IRWs and retrieves the operand at the end of the list. To access operands in stacks other than the currently active one, a modified form of the IRW—the stuffed IRW—is used.

Program references to data items not on a stack are always through descriptors similar to the segment descriptor described above. On the first access via these descriptors, the corresponding data item is loaded into main storage and its absolute address is stored in the descriptor. Future accesses via this descriptor are efficient, since the absolute address of the data item is available. Because descriptors are specially tagged, user programs cannot manipulate them to obtain absolute addresses. This fact is vital in enforcing protection.

Addressing in the B7700 can be done, therefore, in three ways:

- implicitly, to the stack top,
- explicitly, using IRWs and SIRWs, to all other locations on stacks, and
- via descriptors to areas of memory not part of any stack.

The B7700's instruction repertoire includes most of the operations possible on contemporary computers: fixed- and floating-point arithmetic, data transfer from and to stacks and between operands not on stacks, editing and data conversion, program control, and so on. The presence of tag bits makes it easy for the hardware to detect invalid operations—the use of uninitialized operands, for example.

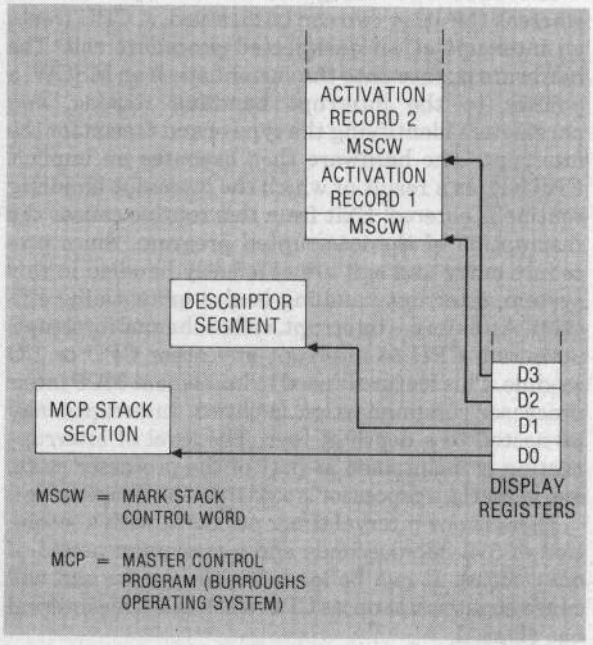


Figure 15. Stack structure and display registers in the Burroughs B7700.

Procedure entry and exit are extremely efficient. To enter a procedure, a user program pushes onto the stack an MSCW, the identity and starting address of the procedure being called, and any parameters to be passed. An ENTER operator is then executed. The hardware uses the contents of the stack top to perform procedure linkages and sets up the display registers appropriately. Procedure exit is accomplished by reversing these steps.

A variety of mechanisms provide protection. First of all, a limit-of-stack register and base-of-stack register are set for the currently active stack, to prevent stack overflow and underflow. Every descriptor contains the size of the data item it refers to; each memory access via the descriptor is checked to ensure that only memory locations within the corresponding data item are being accessed. Tag bits attached to each word provide a third dimension of protection, since invalid operations can then be detected. As in all other computer systems discussed in this survey, the B7700 has two states of operation—normal (in which user programs execute and in which certain operators cannot be executed) and control (in which all operators are valid).

The B7700 has a comprehensive interrupt scheme which fits in neatly with its stack-oriented architecture. Interrupts fall into four categories, listed below in descending order of priority:

- Alarm, e.g., memory parity errors, processor hardware errors;
- Syllable, e.g., invalid program operations, arithmetic overflows;
- Special, e.g., timer interrupts, stack overflow; and
- External, e.g., interrupts by I/O modules or other CPUs.

The first two categories are always enabled, whereas the other two can be disabled. A CPU treats an interrupt as an unexpected procedure call. The hardware pushes onto the current stack an MSCW, a pointer to the interrupt handling routine, and parameters identifying the type of and reason for the interrupt. The hardware then executes an implicit ENTER, as a result of which the interrupt handling routine is entered. Exit from this routine causes the resumption of the interrupted program. Since procedure entry and exit are efficiently handled in this system, interrupt handling is also (potentially) efficient. An external interrupt, called channel interrupt, permits a CPU to interrupt any other CPU or I/O module. This feature is used to implement MCP interprocessor communication facilities. Interrupts may be nested to a depth of four. The level of interrupt nesting is maintained as part of the processor state, and at level i, a processor is said to be in control mode i.

There is one interval timer per CPU, with a resolution of 512 microseconds and a maximum period of one second. It can be loaded with a value and will cause an interrupt on its CPU when this time interval has elapsed.

In all the cases described above, the CPU to be interrupted is decided by the originator of the interrupt. There is no competition among CPUs for interrupt servicing, as is the case in the Univac 1100/80, for example.

Memory modules. The memory modules in the B7700 not only physically house memory but also contain logic to perform the functions of storage control and conflict resolution. There are eight ports per memory module, and one requester can be connected to each port. Each port is assigned a priority level, which is used by the memory module to resolve memory request conflicts. I/O modules are given the highest priority so that latency times can be minimized for I/O devices trying to access memory. The conflict resolution algorithm used by the memory module guarantees starvation-free service to all requesters.

Each memory module can be two- or four-way interleaved. The address range corresponding to each memory module is set in memory limit registers in that memory module. These registers can be set under program control. Each requester's main memory interface has the contents of all limit registers available to it; this information is used to decide which memory module a memory request should be routed to.

A request inhibit register is present in every memory module and has a bit corresponding to each requester. The register can be read or written under program control by all CPUs. If bit i is set to 1, memory requests from requester i are ignored by that memory module. This feature is useful in error recovery, where a requester suspected of being faulty can be isolated under program control.

I/O modules. The I/O module in the B7700 performs a function that can best be described as being midway between that of a peripheral processor (as in the Cyber 170) and that of a channel (as in the IBM 370). Like the latter, the I/O module is not programmable and executes commands set up by a CPU. However, it does not have to interrupt a CPU after completion of an I/O operation; like a peripheral processor, it can service a number of requests before signaling its completion.

Crucial to the I/O scheme is a data structure located in main memory called the I/O subsystem map. This map's location and substructures are defined for all I/O modules at system initialization. All requesters in the system share this single map, which contains information such as device status, paths available to each device, etc.

When MCP receives a program's I/O request, a data structure called an IOCB—input-output control block—is created and linked onto the I/O queue for the appropriate device. If an I/O module is currently servicing this queue, nothing more needs to be done by the CPU. Otherwise, when a reasonable number of requests have accumulated in the queue, a CPU writes a start I/O command for the device into the I/O map and interrupts the device. The I/O module then reads the command and begins processing the first IOCB on the queue. On completion, the IOCB is de-

linked from the original list and placed in a queue called the status queue. The I/O module then begins processing the second request and continues in this manner until it exhausts the queue, encounters a fatal error, or is interrupted by a CPU. The removal of IOCBs from the status queue can be done by any CPU at any time. Concurrent manipulation of different parts of the I/O map by different requesters is possible; each substructure has lock bits to ensure mutual exclusion in critical regions.

I/O errors, unless internal to an I/O module, do not cause the I/O module to signal an interrupt. An IOCB for which an I/O error has occurred is treated just as if it were successful—it is delinked and placed in the status queue. However, the I/O module does place diagnostic information specifying the nature and cause of the error in the offending IOCB.

When more than one path is available to a device, an I/O module uses information from the I/O map to choose an appropriate path and to try alternate paths if the first is busy. Besides the standard method of performing I/O (described above), special I/O commands are available and are used, for instance, in system initialization, error recovery, and other such situations. I/O can also be done on an interrupt-driven rather than a queue-driven basis if a CPU so desires.

The master control program. MCP for the B7700 is a compatible version of MCPs for earlier Burroughs machines and supports both multiprogramming and multiprocessing. Jobs execute in a segmented virtual address space and share programs with other jobs. Multitasking is possible; each time a job creates a subtask, a new branch of the current task's stack is created. The original task and subtask share the stack contents up to this point, but have individual, non-shared branches above. Logically, however, each sees a monolithic stack. This procedure may be repeated recursively, yielding a "cactus stack" (Figure 16).

User programs use MCP facilities via procedure calls to appropriate routines located in the MCP stack section. Unlike most other systems, which have to execute a special trap instruction for calls to the operating system (e.g., the SVC instruction in the S/370), the B7700 uses the same ENTER operator for calling both user and MCP routines.

Communication facilities between tasks are implemented by means of software interrupts. A task may ask to be interrupted by MCP if a certain event, usually caused by some other task, occurs. This feature is extremely useful in synchronization and coordination of subtasks within a job. Software interrupts appear as unexpected procedure calls to the corresponding interrupt service routines. While awaiting an event, a task may choose to either continue processing or enter a blocked state—MCP provides both options.

MCP has extensive error recovery facilities. Dynamic system reconfiguration is possible without reinitialization. I/O modules or CPUs can be effectively isolated by setting the requester inhibit registers appropriately. The possibility of recovery from software errors within MCP is greatly enhanced because each control mode (from CM_1 to CM_3) has a different MCP stack section and hence different interrupt handling routines (the hardware being aware of the location of each such stack section). If an error occurs in an MCP interrupt handling routine, the ensuing interrupt can be routed to a different interrupt handling procedure, even if this interrupt is of the same type as the first one. This can be recursively repeated up to three times. Hence MCP can limit the propagation of software errors, since different interrupt handling procedures are used in each control mode. The presence of multiple paths to a device permits MCP to continue operations on that device even if one of the paths becomes unusable.

MCP is written almost entirely in a high-level language called Espol, a Burroughs dialect of Algol. All user programs are also written in high-level

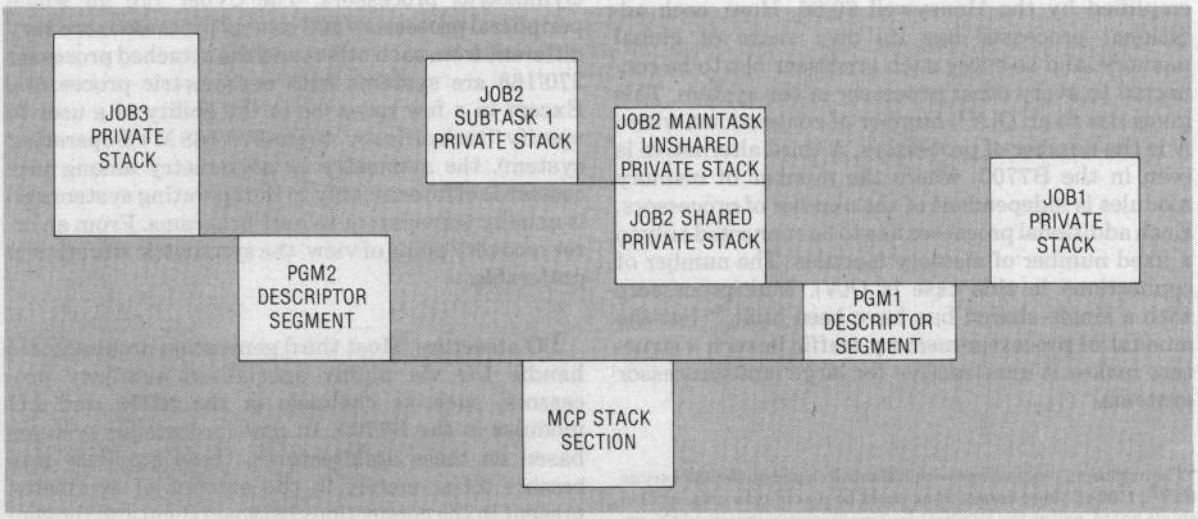


Figure 16. The "cactus-stack," a shared stack structure in the Burroughs B7700.

languages. Since code generated by the compilers does not contain sensitive or potentially malignant operators, such operators do not need to be marked as privileged. This accounts for the B7700's extremely small privileged instruction set. As mentioned earlier, all programs are reentrant, and code sharing is done as much as possible.

General observations

Having studied five multiprocessing systems in detail, we can now look back and see if we can find any common threads running through the designs of these systems. Although multiprocessing systems give rise to many problems, they also offer many opportunities which, if successfully exploited, can make multiprocessing worthwhile.

Interconnections in a multiprocessor. There are two basic perceptions of the relationship between uniprocessors and multiprocessors. One views a multiprocessor as a number of connected uniprocessors. The other thinks of a uniprocessor as a degenerate case of a multiprocessor. In a simple sense, the first is the worm's eye view—and the second the bird's eye view—of multiprocessing. It is not surprising that the designer's choice of a connecting scheme depends greatly on which of the two views he takes. The IBM 370/168 and the Honeywell 60/66 are examples of the first viewpoint, while the Burroughs B7700 and C.mmp[36,37] are instances of the second.

The most important connections in a multiprocessor are those between the individual processors and the shared memory. The conceptually simplest way to make these connections (though certainly not the most common) is to use a crossbar switch as in the C.mmp. This requires the switch to have one connection from every processor and memory module. The total number of such connections will therefore vary linearly according to the number of processors and memory modules.* An alternate way to connect memory to processors is exemplified by the Honeywell 60/66. Here, each additional processor has its own share of global memory, and so every such processor has to be connected to every other processor in the system. This gives rise to an $O(N^2)$ number of connections, where N is the number of processors. A third alternative is seen in the B7700, where the number of memory modules is independent of the number of processors. Each additional processor has to be connected only to a fixed number of memory modules. The number of connections in this case is $O(N)$. Multiprocessors with a single shared bus have been built,[34] but the amount of processor-memory traffic in such a structure makes it unattractive for large multiprocessor systems.

*The number of connections within the switch will obviously vary as $O(N^2)$. But all these connections could be placed on a chip, or on a single board, and so the number of external connections would be only $O(N)$.

Besides connections to shared memory, a multiprocessing system usually has two other types of connections. One is a connection among processors to facilitate interprocessor communication. The Univac 1100/80 and the Honeywell 60/66 have one such connection between every pair of processors, giving an $O(N^2)$ number of connections. The B7700, however, has a single interrequester interrupt bus for the entire system. Every requester is connected to this bus and shares it with every other requester. While bus sharing does introduce problems such as contention and delays due to bus arbitration logic, the frequency of interprocessor communications is usually low enough to make the single shared bus feasible.

Most of the systems studied here have a cache associated with each processor. In order to ensure the consistency of (possibly) multiple copies of data in the different caches, every cache must communicate with every other cache. Systems such as the Univac 1100/80 and the Honeywell 60/66 have a cache invalidate interface between every pair of caches, thus requiring an $O(N^2)$ number of connections. A *shared* cache invalidate bus could reduce the number of interconnections, although the high bandwidth such a bus would require probably explains why no multiprocessing system uses such an approach.

To correctly route memory accesses to the appropriate memory modules, a number of systems have one or more coordinating elements. The crossbar switch in the C.mmp, the system controller in the Honeywell 60/66, the system interface unit in the Univac 1100/80, and the multisystem control unit in the IBM 370/168 are examples of this. However, such coordinating elements are not essential, as demonstrated by the B7700, where each requester does its own routing of memory and interrupt requests.

Symmetry. The processors used in a multiprocessing system may or may not be identical. The Honeywell 60/66, the Univac 1100/80, and the true multiprocessor 370/168 are systems having identical (symmetric) processors. The Cyber 170 (in which peripheral processors and central processors are very different from each other) and the attached processor 370/168 are systems with asymmetric processors. Except in a few cases (as in the ability of a user to specify "task affinity" to the 370/168 MVS operating system), the symmetry or asymmetry among processors is of interest only to the operating system and is usually transparent to user programs. From an error recovery point of view, the symmetric situation is preferable.

I/O structure. Most third generation architectures handle I/O via highly specialized auxiliary processors, such as channels in the S/370 and I/O modules in the B7700. In multiprocessing systems based on these architectures, these auxiliary processors differ mainly in the amount of symmetry present in the connections between them and the central processor.

In a fully symmetric situation, there is no a priori correspondence between the central processor(s) and the auxiliary processors. Each auxiliary processor can communicate with any central processor, and each central processor can do I/O via any auxiliary processor. Such a structure is found in the B7700, in which central processors and I/O modules are treated equally as "requesters of memory," with every requester able to communicate with every other requester.

The diametrically opposite approach is found in systems where there is a permanent correspondence between auxiliary processors and central processors. In the 370/168, for example, every channel is permanently associated with a CPU, and a CPU cannot do I/O via a channel attached to another CPU.

Between these extremes one can have I/O structures with varying degrees of symmetry. The maximally configured Univac 1100/80, for example, has two storage interface units, each attached to two central processors and two I/O modules. An I/O module attached to one storage interface unit cannot communicate with a CPU attached to the other storage interface unit. Both CPUs attached to a storage interface unit can, however, communicate with and perform I/O on both the I/O modules connected to the same storage interface unit.

For numerous reasons, the fully symmetric situation is the most desirable. First, a system with a fully symmetric I/O structure can continue operating in spite of failure in a central or auxiliary processor. In an asymmetric situation, failure in a central processor disables the auxiliary processors associated with that central processor. Second, interrupt latency times (i.e., the time between the sending of an interrupt signal by an auxiliary processor and its being accepted by some central processor) are likely to be less in the fully symmetric case, mainly because the probability of all central processors being disabled for interrupts is much less than the probability of any one central processor being disabled. Finally, task switching and interprocessor communication overheads are likely to be much smaller in a fully symmetric situation. In an asymmetric situation, a program's request to do I/O via an auxiliary processor can only be handled by the central processor associated with that auxiliary processor. This can cause significant overheads in task switching and in communications between the central processor running the program and the central processor servicing the I/O request.

Interrupts. Interrupts can be classed into two types: internal interrupts, such as arithmetic overflows, divide checks, and protection violations; and external interrupts, such as I/O interrupts and timer interrupts. In all the systems discussed here, internal interrupts are handled on the processor on which they originated. The systems vary, however, in their handling of external interrupts, the major difference being the extent to which interrupts are shared by the processors. In the Honeywell 60/66, all interrupts are routed to one processor labeled the "control processor." While probably the simplest to implement in hardware, this scheme suffers from poor recoverability and long latency times, as discussed earlier.

In the Univac 1100/80, all external interrupts (except I/O interrupts) are shared by all the processors. Each interrupt is given to every processor, one after the other, for a predefined time period. The first processor to respond during its time slot gets the interrupt. Such a scheme obviously will have excellent recoverability and short latency times.

A variation of the previous scheme is to let the interrupter choose which processor it wishes to interrupt. The SIGP interrupt in the S/370 and the interrupt mechanism used by the I/O modules in the B7700 fall within this category. To overcome the problem of long latency times, the interrupter could change its target if the original target did not respond within a reasonable time. From a recoverability point of view, this scheme is as good as the previous one.

Interprocessor communication mechanisms. There are a number of reasons why a hardware interprocessor mechanism is necessary in a multiprocessing system. If the structure of the system is asymmetric, there will be frequent service requests exchanged between the different processors. A hardware interprocessor communication mechanism, to draw the attention of the target processor, would be very useful in such cases. The mechanism also facilitates synchronization between processors. For instance, during a checkpoint all processors except the one executing the checkpoint task could be halted. An interprocessor communication facility would permit such information to be easily and quickly broadcast throughout the system. Also, in the event of failure, a hardware-initiated signal to all functioning processors would inform them of the failure.

Since processors share memory, it is clearly possible to have software communication without explicit hardware assistance. Doing that, however, introduces a number of problems. First, all processors will have to periodically check to see if there is a message for them. Second, since a processor only recognizes requests when it does its next polling of messages, response times to requests can become intolerably long. Further, if a processor fails when a job holding locks is running on it, and if there is no way for other processors to be aware of this, a deadlock can occur.

Interprocessor communication mechanisms have various levels of sophistication. The Cyber 170 doesn't have such a mechanism at all. The Univac 1100/80 and the Honeywell 60/66 have a simple, software-initiated interprocessor interrupt. The S/370 has a more sophisticated mechanism, the SIGP, which enables one processor to act virtually as an operator for another processor. The SIGP also has the unique, hardware-initiated emergency malfunction, discussed earlier.

Reserved storage areas. In most of the architectures discussed here, certain areas of main storage are reserved for special use by the hardware. In the

S/370, for instance, addresses 0-4K contain data areas related to I/O, the interrupt vectors (i.e., the old and new program status words for all interrupt classes), and areas used by the hardware to record diagnostic information when failure occurs. In a multiprocessing situation, there are a number of processors, each with a set of reserved addresses; this leads to a potential clash of reserved areas, one which can be resolved in three ways.

One method—used in the Univac 1100/80—makes the physical storage areas corresponding to the reserved addresses the same in all the processors. The Univac 1100/80 allows this because the reserved area mainly contains interrupt vectors; each interrupt handling routine therefore has to either be reentrant or have locks that ensure mutual exclusion.

The Honeywell 60/66 solves the problem in a different way: the address of the reserved storage area is different for each processor. As a result, the physical storage locations corresponding to the reserved storage addresses is different in each case.

In the S/370, the processors treat addresses 0-4K as reserved addresses. However, the physical storage corresponding to this address range is different for each processor. This is accomplished by the previously mentioned "prefixing" mechanism.

The last two techniques achieve the same effect, and consequently there is really little to choose between them.

The operating system. Surprisingly, in spite of the large differences in architecture and system organization, the operating systems discussed here are similar in the way they handle multiprocessing. In all cases, the operating system screens users from the specific hardware configuration—as far as they are concerned, the system is a uniprocessor whether there are in fact two, three, or more processors. Each user task can create independently schedulable subtasks, which compete for resources with the parent task. The operating system normally provides the user with software mechanisms to synchronize tasks and to permit communication between them. Locking mechanisms (built using indivisible hardware instructions such as the test-and-set or compare-and-swap) are provided to permit strictly serial access to critical sections of shared code and data. There is usually only one copy of the operating system, with different parts reentrant or serially reusable. Since all these operating systems support multiprocessing, the usual issues arising in a multiprogramming context are also present: address space management, scheduling, protection, etc.

Probably the two greatest sources of variation among the operating systems are their locations and the relationships they set up between the processors. In the Cyber 170, for example, the operating system is executed by one peripheral processor, P0. All the other processors, whether central or peripheral, are treated as slaves to P0. A similar situation, involving symmetric rather than asymmetric processors, is found in the DECsystem-10.[33] Here, there are two identical central processors, one designated as master and the other as slave. The operating system runs only on the master, with the slave treated just as a schedulable resource. The most common approach, however, is a distributed operating system, in which all processors may execute parts of the operating system as necessary. MVS[35] on the S/370 and Hydra[37] on the C.mmp are examples. This approach is certainly the most symmetric, the most immune to individual processor failure, and the most aesthetically pleasing.

Error recovery. In the introduction I mentioned that one advantage of a multiprocessing system is its superior error recovery capability. Note that recoverability is not synonymous with reliability. Because of the interconnections and extra units present, the probability of failure may in fact be higher on a multiprocessor than on a uniprocessor. On the other hand, the inherent redundancy of a multiprocessor probably increases its ability to carry on in spite of failure. The mechanisms assisting error recovery span a wide range. The S/370 has a hardware-generated malfunction alert, which informs all other processors of a failure in one processor. The Univac 1100/80's auto recovery timer assists in recovery from software crashes and transient hardware failures. The B7700's requester inhibit registers permit recovery software to logically isolate faulty requesters.

Even with such features, software for error recovery is likely to be complex. For example, saving a program on a failed processor involves backtracking the execution of the program to a point before intermediate results were corrupted by the failure, and continuing execution from there. This can be difficult, especially in a processor with a pipelined execution unit. Another problem is that the program on the failed processor may be holding some locks. The error recovery procedure has to be smart enough to ensure that deadlocks are not introduced as a result of the recovery process itself!

Performance. As stated earlier, there is little quantitative data on the performance of the computer systems surveyed. However, one can make some general statements about the performance of multiprocessing systems.

Invariably, the scaling in performance of a multiprocessing system is sublinear. That is, using an N-processor multiprocessing system always yields less than N times the performance of the corresponding single-processor-based system.*

Contention among processors for memory and delays involved in ensuring cache validity are two

*This statement is strictly true only if "performance" means "maximum performance." All parameters other than the number of processors (amount of storage, I/O configuration, etc.) should be adjusted in each case for optimal performance. Not doing so can yield anomalous results, indicating that a multiprocessor does indeed have better performance than the sum of the performances of its processors running as uniprocessors. Stated more accurately, therefore, a multiprocessor's maximum performance is always less than the sum of the maximum performances of its corresponding uniprocessors.

major factors in this performance degradation. Besides these hardware overheads, a multiprocessor has certain unique software overheads. Scheduling is more complex because of the presence of more than one processor, and this can contribute significantly to the extra overheads. Locks are required in a multiprocessor, although they can be avoided in a uniprocessor by disabling interrupts whenever critical sections of the operating system are being executed. The time spent waiting at these locks contributes to the loss in performance. Finally, the greater computational power of a multiprocessor implies a correspondingly higher level of multiprogramming. At high levels of multiprogramming, certain "large system effects" are felt—queues get longer, hash tables become fuller, certain data structures become too large to be stored in main memory, etc. All these factors cause multiprocessor performance to obey the law of diminishing marginal utility—i.e., each added processor contributes relatively less to overall performance. ■

Bibliography

IBM 370 Model 168

1. Arnold, J. S., et al., "Design of Tightly-Coupled Multiprocessing Programming," *IBM Systems J.*, Vol. 13, No. 1, 1974, pp. 60-87. A good description of the components of MVS that play a role in multiprocessing. The sections on alternate CPU recovery, CPU affinity, and locking are particularly interesting. This paper is worth reading, at least to gain an appreciation of the difficulties involved in designing an operating system for multiprocessor environments.
2. Case, R. P., and A. Padegs, "Architecture of the IBM System /370," *Comm. ACM*, Vol. 21, No. 1, Jan. 1978, pp. 73-96. A readable account of the architecture of the S/370 and of its evolution from its predecessor, the S/360. Includes a section on multiprocessing. Highly recommended.
3. Brinch Hansen, P., *Operating System Principles*, Prentice-Hall, Englewood Cliffs, N.J., 1973. Section 3.5 discusses deadlocks and their prevention. MVS uses the linear ordering technique described in that section.
4. IBM Corp., *IBM System/370 Principles of Operation*, Form no. GA22-7000-5, Aug. 1976. The definitive work on the S/370 architecture. Reads at times like a legal document, but is unsurpassed in its precision and unambiguity. Includes a separate chapter on multiprocessing.
5. IBM Corp., *IBM System/370 Model 168 Functional Characteristics*, Form no. GA22-22-7010-4, Jan. 1976. Gives information specific to the 370/168. Details such as processor speed, channel data rates, and maximum memory available can be found here. Appendix E, exclusively devoted to multiprocessing, describes reconfiguration controls.
6. MacKinnon, R. A., "Advanced Function Extended with Tightly-Coupled Multiprocessing," *IBM Systems J.*, Vol. 13, No. 1, 1974, pp. 32-59. Describes the S/370 hardware features used in multiprocessing. Includes comparison of past and present multiprocessor systems supported by IBM.
7. Scherr, A. L., "OS/VS2-2 Concepts and Philosophies," *IBM Systems J.*, Vol. 12, No. 4, 1973, pp. 368-381. Discusses the structure and design of MVS.

Control Data Cyber 170 Series

8. Atwood, J. W., "Concurrency in Operating Systems," *Computer*, Vol. 9, No. 10, Oct. 1976, pp. 18-26. Has a good description of Kronos. The structure of the operating system and the implementation of various functions are explained well. Quite readable.
9. Auerbach Publishers Inc., "CONTROL DATA CORP. CYBER 170 Series," *Auerbach Computer Technology Reports*, 1977.
10. Control Data Corp., *Cyber 170 Hardware Reference Manual*, Form no. 60420000F, Apr. 1977. Describes the hardware details of the Cyber machines.
11. Control Data Corp., *6000 Series Reference Manual*, Form no. 60100000W, Jan. 1973. Describes the CDC 6000 series, the predecessor of the Cyber 170.
12. Control Data Corp. *SCOPE General Information Manual*. An overview of Scope, CDC's earliest operating system.
13. Control Data Corp., *SCOPE 2 Reference Manual*, Form no. 60342600D, Aug. 1973. Treats Scope in greater detail than the above.
14. Control Data Corp., *KRONOS 2.1 Reference Manual*, Vol. I, Form no. 60407000D, June 1975. Describes CDC's second operating system.
15. Control Data Corp., *NOS 1.0 Reference Manual*, Vol. I, Form no. 60435400A, June 1975. Describes CDC's latest operating system, which supports networking.
16. Thornton, J. E., *Design of a Computer: The Control Data 6600*, Scott, Foresman and Co., Glenview, Ill., 1970. Has an illuminating discussion of the design principles involved in the CDC 6600, much of which is relevant to the Cyber 170. The sections on technology and physical construction are obsolete, but the rest of the book makes good (and worthwhile) reading.

Honeywell Series 60 Level 66

17. Honeywell Inc., *General Comprehensive Operation Supervisor (GCOS)*, Order no. DD19, Apr. 1974. Describes GCOS and its functions. Section I is particularly relevant to our discussion.
18. Honeywell Inc., *Series 60 Level 66 Summary Description*, Order no. DC64, 1975. Gives an overview of the 60/66 and a description of its major hardware and software components. Chapters 1,2,4,8,9,10 are particularly informative.
19. Honeywell Inc., *Macro Assembler Program (GMAP)*, Order no. DD08, July 1974. Section I gives an overview of the hardware. Sections IV and VIII describe the instruction set; Section V details the address modification scheme.
20. Honeywell Inc., *System Operation Techniques*, Order no. DD50 Rev. 1, Oct. 1976. Section VIII describes system reconfiguration techniques.
21. Honeywell Inc., *System Startup*, Order no. DD33 Rev. 2, July 1976. Sections I and II describe the bootstrap loading techniques. Section VI describes the system recovery facilities. Appendix C gives information on hardware reconfiguration.

22. Mitre Corp., *WWMCCS H6000 Multiprocessor Performance Evaluation,* NTIS report no. ADA 039111 (Vols. I & II), National Technical Information Service, Springfield, Va., Feb. 1977. A study of H6000 multiprocessor systems conducted for the Air Force; gives the performance figures and graphs cited in this survey.

Univac 1100 Model 80

23. Borgerson, B.R., et al., "The Evolution of the Sperry Univac 1100 Series: A History, Analysis, and Projection," *Comm. ACM,* Vol. 21, No. 1, Jan. 1978, pp. 25-43. Excellent description of the evolution of the 1100 series. Gives a broader and more general perspective of the Univac 1100 series than this survey; also covers a number of details, such as application software, which have been ignored here since they are not germane to multiprocessing. Highly recommended.

24. Sperry Univac, *1100/80 Systems: Processor and Storage,* Form no. UP-8492. Gives a detailed description of the processor, main storage unit, and storage interface unit; discusses the instruction set, addressing and interrupt schemes, and other hardware details.

25. Sperry Univac, *1100/80 System Hardware (4x4 Capability) System Description,* Form no. UP-8568. An overview of the 1100/80 system, its hardware components, its interconnection scheme, etc. Less detailed level than the above.

26. Sperry Univac, *1100/80 System: System Transition Unit. Programmers' Reference,* Form no. UP-8409. A detailed description of the system transition unit and the program commands available to interrogate and control it. The auto recovery mechanism and the system transition unit's role in it are also discussed.

27. Sperry Univac, *1100 Series Executive, Volume 2, EXEC Level 33R1, Programmer Reference,* Form no. UP-4144.21. Part of a four-volume reference on the EXEC operating system, this volume describes the overall control of the 1100 series by EXEC and is valid for all models of the 1100 series (1100/10 to 1100/80). In refreshing contrast to most computer manuals, this one gives a readable account of EXEC's design philosophy and internal details.

Burroughs B7700

28. Burroughs Corp., *B7700 Information Processing Systems: Characteristics Manual,* Form no. 1059979, 1973. An elementary description of the B7700, its components, and their interrelationships. Includes details such as maximum memory capacity, device speeds, and processor cycle times.

29. Burroughs Corp., *B7700 Information Processing Systems: Reference Manual,* Form no. 1060233, 1973. Encyclopedic description of the B7700. Extremely detailed descriptions of each component. This is one computer manual with too much information rather than too little—one can easily get lost in the details.

30. Chu, Y., *High Level Language Computer Architecture,* Academic Press, New York, 1975. Contains papers on those aspects of computer architecture pertinent to high-level languages. Chapter 4 is particularly relevant, since it discusses stack architectures. This chapter also includes a discussion of tagged memory and of stack structures when parts of the stack are shared (i.e., the "cactus stack").

31. Organick, E. I., *Computer System Organization: The B5700/6700 Series,* Academic Press, New York, 1973. Excellent explanation of how programs are run on the Burroughs architecture. Deals with the predecessors of the B7700, but the material discussed is equally valid for the B7700. Organick's inimitable style makes this book well worth reading.

32. Wegner, P., *Programming Languages, Information Structures and Machine Organization,* McGraw-Hill, New York, 1968. Chapter 4 contains a good discussion of block-structured languages and the run-time support they require.

General observations

33. Digital Equipment Corp., *DECsystem-10 Technical Summary,* Maynard, Mass., 1977.

34. Kraley, M. F., "The Pluribus Multiprocessor," *Digest of Papers—1975 Int'l Symp. Fault-Tolerant Computing,* June 1975, p. 251.*

35. Scherr, A. L., "OS/VS-2 Concepts and Philosophies," *IBM Systems J.,* Vol. 12, No. 4, 1973, pp. 368-381.

36. Wulf, W. A. and C. G. Bell, "C.mmp—A Multi-Mini-Processor," *AFIPS Conf. Proc.,* Vol. 41, Part II, 1972 FJCC, pp. 765-777.

37. Wulf, W. A. et al., "Hydra: The Kernel of a Multiprocessor Operating System," *Comm. ACM,* Vol. 17, No. 6, June 1974, pp. 337-345.

*This digest is available from the IEEE Computer Society Publications Office, 5855 Naples Plaza, Suite 301, Long Beach, CA 90803.

M. Satyanarayanan is currently at Carnegie-Mellon University, working towards a doctorate in computer science. His research interests include computer architecture, operating systems, and the hardware and software of parallel processing systems. A member of the ACM, he has worked for IBM and Digital Equipment Corporation. He graduated with a bachelor's degree in electrical engineering from the Indian Institute of Technology, Madras. He also holds master's degrees in computer science from the Indian Institute of Technology, Madras, and Carnegie-Mellon University.