DEC PDP-11

Tradeoff goals of a mid-range minicomputer design—balancing cost, functionality, and size for broad appeal—are primarily achieved by exploiting technological advances and appraising user needs

Design Decisions Achieve Price/Performance Balance in Mid-Range Minicomputers

J. Craig Mudge

Digital Equipment Corporation
Maynard, Massachusetts

Design evolution of a minicomputer family usually proceeds along three basic dimensions: cost, functionality, and size. That is, the minicomputer becomes cheaper, more powerful, and smaller with time. The underlying hardware technology is the dominant factor in determining the evolution. In contrast to the evolution of large computers, market factors have less influence on the growth pattern of minicomputers. However, minicomputer software characteristics are affected by the market. These requirements rapidly feed down to modify the hardware, given that the technology will support user needs.

The DEC PDP-11/60* serves to demonstrate minicomputer designing with improved technologies. Being a mid-range machine, ie, neither the lowest in cost nor the highest in performance, its design is a rich source of tradeoff examples. Its cache design illustrates a price/performance trade, the decreasing cost of read-only memories (ROMs) show how hardware-microcode tradeoffs change over time, and its integral floating-point arithmetic unit exemplifies a software-hardware tradeoff.

Design Styles

Equipment history reveals that a member is added to a minicomputer family whenever technology advances by a factor of two; for example, doubling of bit density on a memory chip. Over the past 15 years, such an improvement has occurred about every two years.

These advances in technology can be translated into either of two fundamentally different design styles. One provides essentially constant functionality at a minimal price (which decreases over time); the second keeps cost constant and increases functionality. (Here, and in the discussion to follow, the definition of functionality has been broadened from its conventional single component, speed, to include components such as extended instructions and self-checking.) Both design approaches coordinate with the basic marketing philosophy of the minicomputer industry: more computation for more users at less cost. There have been ten models, or implementations, of the PDP-11 architecture since the unit was introduced in 1970.[1] Fig 1 illustrates how the two design styles affected successive implementations within this minicomputer family.

Lower cost members trace the decreasing-cost, constant-functionality curve. (This is the 11/20, 11/05, and LSI-11 or 11/03 line.) The horizontal line in Fig 1 connects the constant-cost, increasing-functionality designs. (Not shown are "growth-path" members that provide greater performance at slightly increased costs; 11/45, 11/55, and 11/70 machines trace an upward

*DEC, PDP-11, and UNIBUS are registered trademarks of the Digital Equipment Corp, Maynard, Massachusetts.

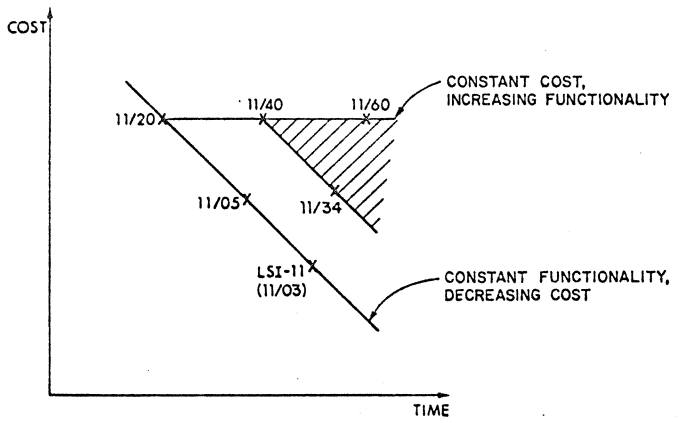


Fig 1 Minicomputer family evolution. Advances in technology translate into two design styles: constant-cost, increasing-functionality and constant-functionality, decreasing-cost. The 11/60 represents former design style. Functionality added to 11/40 is depicted by shaded area. Tradeoffs discussed occur within this area

growth-path curve.) Shaded area in the figure represents the added functionality possible through technology advances. Mid-range minicomputers attempt to optimize price/functionality and, hence, offer an excellent vantage point for discussing design tradeoffs made under the constant-cost design style.

In addition to the capabilities provided by technological advances, a mature family architecture and user base allows the minicomputer designer to include those capabilities that were not considered feasible in the original architecture. These features may not have been included because they were too costly to implement, not sufficiently general-purpose to justify their inclusion, or not perceived as being essential to users. Reliability, maintainability, the integral floating-point unit, and the writable control store (WCS) option represent such capabilities.

Internal structure of the 11/60 (Fig 2) incorporates a 2048-byte cache, memory management unit (for virtual-to-physical address translation), and an integral floating-point unit as standard components. The unit can perform a register-to-register add instruction in an average time of 530 ns; internal cycle time is 170 ns. Available as options are a floating-point processor, which implements at higher speed the same 46 instructions as the integral unit, a writable control store, and a microdiagnostic unit.

Advances in Memory Technology

Improvements in memory technology have been the principal forces in minicomputer developments. Memory

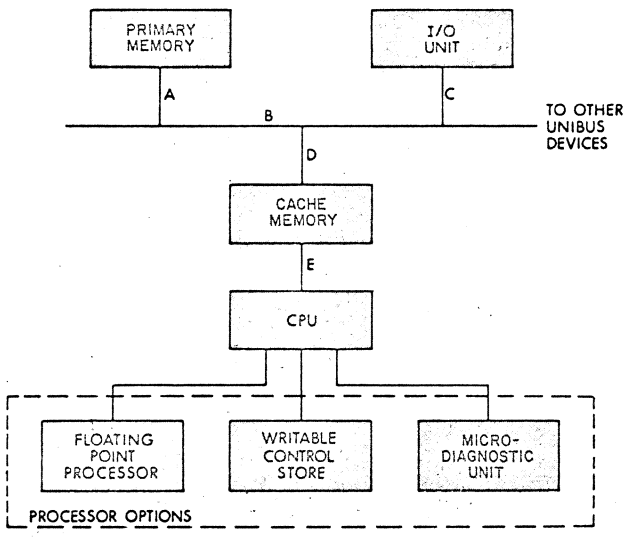


Fig 2 Internal structure. Cache placement between Unibus and CPU permits faster execution and allows use of standard memories. However, DMA monitoring mechanism is needed for traffic on path CBA. Module count is six for CPU and cache, one for writable control store, one for microdiagnostics unit, and four for floating-point processor. This processor operates in parallel with CPU execution of nonfloating-point instructions; instruction times are 1.02 μs for double-precision add and 1.53 μs for single-precision multiply. Writable control store uses 1024 control words that are reloadable and that control 170-ns inner machine. Machine is design optimized for user environment characterized by real-time operating system and FORTRAN

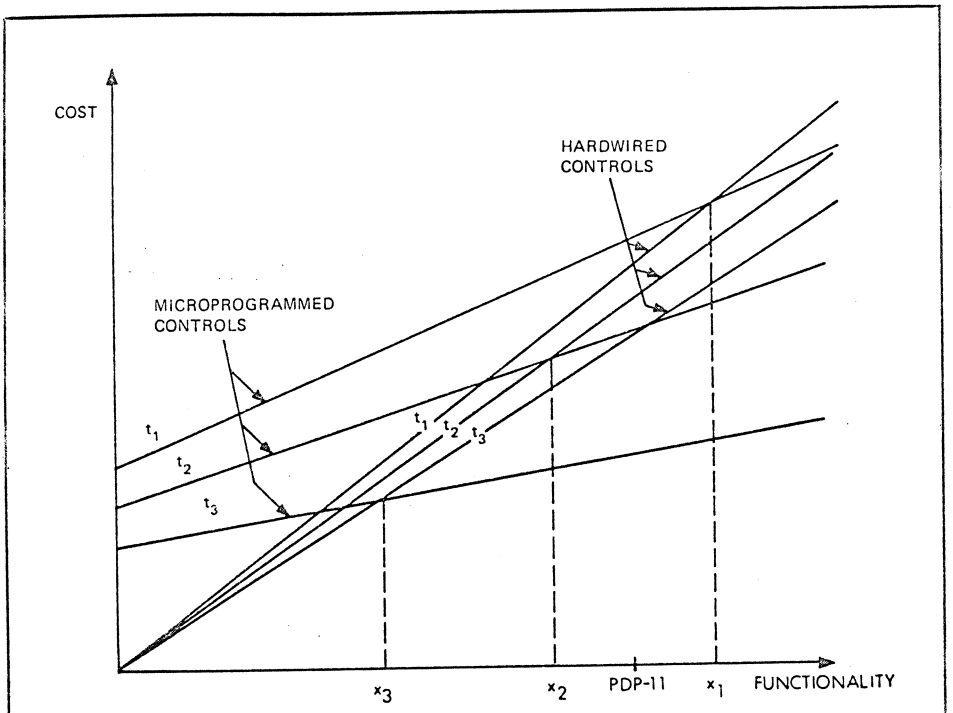


Fig 3 Semiconductor technology trends in control implementations. Cost comparisons, at three different points in time, of conventional hardwired control and advanced microprogrammed control show two important trends. First, at fixed point in time in 1970s (eg, time t3), microprogrammed control is less expensive above certain level of complexity (x3). For simplest type of machine, random logic gives most economical design. Microprogrammed design has base cost associated with address sequencing and memory selection circuitry. Microprogrammed control cost increases slowly with number of sequencing cycles, which are added as complexity increases, because each additional cycle requires one additional word of control store. Second, because rate of cost-decrease for memories is greater than the rate for random logic, cross-over points move with time, gradually shifting in favor of microprogrammed control. When 11/20 was designed (time t1) hardwired controls were cheaper. Its successor, the 11/40, was designed at time t2 and used microprogramming. The 11/60, at time t3, used increased microprogramming

is the most basic component of a computer and it is utilized throughout the design. In addition to obvious uses as main program and data memory, and as file storage devices (discs and tapes), memory is also located within the central processor in the form of registers, state indicators, control, and buffer storage between the central processor and main (primary) memory. In input/output (I/O) devices, there are buffers and staging areas. Memory can be substituted for nearly all logic by substituting table lookup for computation.

The constantly increasing bit density mentioned previously has been the most dramatic development in memories. For example, bipolar read-write or random-access memory (RAM) chips have advanced as follows:

Year When First Widely Available	Number of Bits
1969-70	16
1971-72	64
1973	256
1975	1024
1977	4096

Cost reductions have paralleled bit density increases. A consequence of higher density RAM technology is that cache memories are now extensively used in mid- and upper-range minicomputers. Bipolar ROM densities have led RAM densities by about a year. Thus, the 2048-bit ROM, organized as 512 x 4, was available in 1975.

These factors have made microprogrammed control increasingly attractive to the minicomputer designer. While large scale computers utilized extensive microprogramming during the 1960s, it was not a cost-effective choice for the minicomputer designer because of the prohibitive cost of the read-only storage technology then available.

Both hardwired control devices and microprogrammed control devices have curves that trace increases in cost as they implement increasing functionality (see Fig 3). However, the rate of cost increase is less for microprogrammed controls than for hardwired controls. Davidow[2] demonstrates that a factor-of-four difference exists between the two slopes.

At some point, the two related hardwired and microprogrammed curves cross. Beyond that intersection, mi-

croprogrammed controls are more economical to use in a design. Both of these curves are moving downward in cost with time, but the curve for microprogrammed controls is moving downward at a faster rate. Thus, the intersection point of the two curves is gradually shifting in favor of microprogrammed controls because the two technologies are moving at different rates. The PDP-11 family offers an example of this trend. At the time the 11/20 was designed, the crossover point was to the right of the PDP-11 instruction set on the abscissa. Hence, the 11/20 used hardwired controls. However, all subsequent implementations have used a ROM-controlled microprogrammed processor. O'Loughlin[3] contrasts the control implementations of four members of the family.

Instruction decode on the 11/60 provides an example of a different use of ROMs. For the secondary decode (the primary is done by combinational logic), part of the instruction register addresses a ROM in which control-store-address offsets are stored. This data-table approach offers both a component saving and a more systematic design. Another example is a ROM-stored table that inspects memory addresses to detect those which refer to locations internal to the processor.

Other advances in semiconductor technology that have affected the minicomputer designer's task include the development of 3-state logic devices and greater levels of gate integration in logic chips. Widely available in 1975, 3-state logic encourages bus-oriented designs. Six 3-state buses are implemented in the 11/60. Examples are the 48-bit wide control signal bus in the CPU and the 60-bit wide fraction data and 10-bit wide exponent data buses in the floating-point processor.

Increased gate integration in logic chips had its major impact on constant-cost minicomputers when the design evolution moved from the 11/20 to the 11/40. The latter machine made heavy use of medium scale integration (MSI). The MSI available to 11/60 designers had negligible density gains over that available to the 11/40 designers. However, after the basic technology decision for the 11/60 was made, a significant step in gate integration did occur. The bit-slice technology, as typified by the 4-bit wide bipolar AM2901 microprocessor slice, became widely available. A 1977 technology decision for a mid-range minicomputer would clearly choose bit-slice components. For the 11/60, however, improvements came from the introduction of 3-state logic and from availability of a wider range of Schottky logic components.

Three semiconductor technology advances contributed to the 11/60 price/performance design in differing degrees. Most important was the cost reduction in ROMs; next was the density improvement in RAMs, and third was the (minor) increase in random logic density.

Price/Performance Balance

Two components, the cache memory and the medium-bandwidth I/O structure, demonstrate the price/perfor-

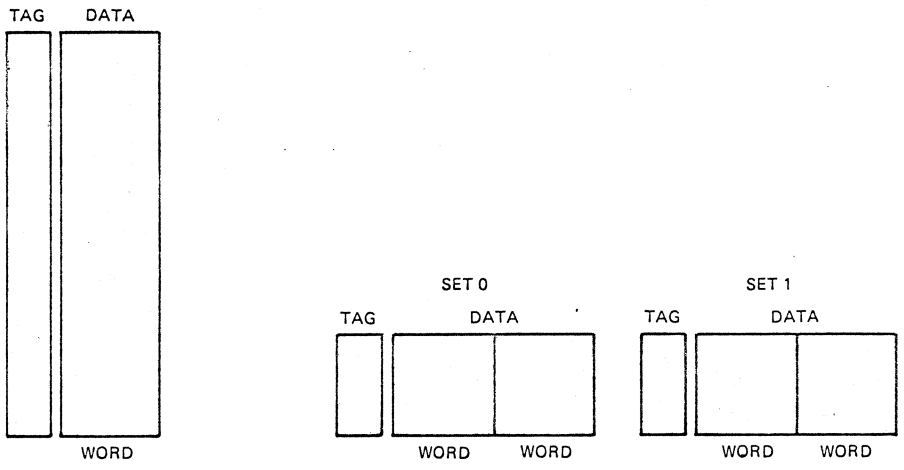


Fig 4 Cache comparison. Simple direct mapped cache of the 11/60 contrasted with the 11/70 cache illustrates a price-performance tradeoff. 11/70 cache has a block size of two (two words are transferred from primary memory) and a set size of two (a word may be placed in either set). Component savings of the simpler organization are clear; only one address comparator is needed, no multiplexer is required to select the output of the data store, and only one set of parity checkers is needed. Hit ratio of the simpler 11/60 cache is 0.87 as compared with 0.93 for the 11/70 cache, which required five times the component count

mance balance characteristic of the 11/60 mid-range minicomputer.

Cache is now a well-proven technique in computer memory implementation. Its purpose is to achieve the effect of an all high speed memory by using two memories—one slow (primary) and one fast (cache)—and by taking advantage of the fact that most of the time, data being used are in the fast or cache memory. Programs typically have the property of locality; that is, over short periods of time, most accesses are to a small number of memory locations. The hardware algorithm managing the cache attempts to keep copies of these locations in the cache. The term "hit ratio" is used to describe the proportion of requests for data or instructions which are satisfied by reference only to the cache. Alternatively, "miss ratio" is the complement of hit ratio. Performance is determined by the hit ratio, which is a function of several cache organizational parameters, including (a) cache size, (b) block size (amount of data moved between the slow or primary memory and the cache), and (c) form of address comparison used.

Strecker[4] describes the research which led to the use of a cache memory in the 11/70. His simulation models were also used in the 11/60 design. By comparing the designs of these machines, several tradeoffs made to obtain a lower cost memory system appropriate to the mid-range 11/60 can be noted.

First parameter to be determined was the amount of data to be moved between primary memory and cache. This decision was closely related to the width of the internal memory bus connecting I/O devices to primary memory. Since the 11/70 was planned to support several high speed direct memory access (DMA) devices, eg, swapping discs operating concurrently, its designers provided a 32-bit bus to memory to supplement the 16-bit wide Unibus. Because the target 11/60 users do not require such a large I/O bandwidth, the Unibus is used for DMA traffic. The 11/70 cache has a block size of two 16-bit words and transfers 32 bits from memory to cache across its dedicated memory bus. Since the 11/60 uses the 16-bit Unibus as its memory bus, the simplest block size—one 16-bit word—was chosen. Note that a 2-word block size can be achieved with a 16-bit bus; the bus is cycled twice to effect a 2-word transfer. Cache simulations showed that this bus cycling would raise the hit ratio of the 11/60 from 87% to 92%. However, the associated performance gain was judged not to be worth the significant added cost of the extra control logic needed to cycle the bus twice.

The next decision concerned the size of the cache. Simulation results showed that the miss ratio decreases rapidly for cache sizes up to 1024 words and less rapidly for larger sizes. But how should the 1024 words be partitioned? Because a full-associative cache requires an expensive content-addressed memory, the partitioning choice for minicomputers is that for a set-associative cache. Since a complete discussion of associativity and replacement is beyond the scope of this article, the reader is referred to the papers by Meade[5] and Strecker[4].

Degree of associativity and total cache size were dominated by the form factors of two candidate RAM chips (256 x 1 and 1024 x 1). These factors are illustrated in Fig 4. The following list shows the clear price/performance advantage of the chosen 1024-word, set-size-of-one cache.

RAM Chip Capacity	Set Size	Cache Size	RAM Chip Count	Hit Ratio
256 x 1	1	256	n	0.70
256 x 1	1	512	2n	0.75
256 x 1	2	512	2n	0.82
1024 x 1	1	1024	n	0.87
256 x 1	2	1024	4n	0.93
1024 x 1	2	2048	2n	0.93

Resulting structure is shown in Fig 5. This simple, direct-mapped organization should dominate minicomputer cache designs in the near-term future. By using the design evolution model shown in Fig 1, it is projected that the two candidate RAM chips for the successor to the 11/60 cache will be the 1024 x 1 and 4096 x 1 chips. Obviously, the design choice for that new class of machine will be a 4096-word direct-mapped cache.

Since simulation data show negligible performance difference between various write-allocation strategies, the lowest-cost strategy, that of allocate-on-write, was implemented. Because the 11/60 utilized a set-size-of-one cache, there was no need to decide upon a replacement algorithm. The 11/70 uses a random-replacement algorithm.

Next decision to be made concerned placement of cache. Two choices were evaluated. The cache could be placed between the Unibus and the primary memory, or between the Unibus and the central processor. The latter was chosen because of these advantages:

(1) Machine execution is faster since the high speed cache is local to the central processor. Time delays associated with synchronization and transmission on the Unibus are avoided.

(2) Instead of designing 11/60-specific memory modules, existing memory subsystems which interface to the Unibus could be used. Moreover, as faster Unibus-interfaced memories become available, they can be installed on the machine without change.

(3) DMA traffic interferes with processor activity to a lesser extent. DMA activity takes place over the path labeled ABC in Fig 2. Processor speed is degraded by interference with I/O operations only when the cache needs to reference the primary memory, using path ABD in Fig 2. This happens only in the event of a read miss, typically less that 13% of the time, and on write operations (10% of memory references).

Disadvantage of this placement is that a mechanism to monitor DMA traffic must be added to the cache to avoid the "stale data" problem. (When the processor reads a location which has been written by DMA, it must receive the information from primary memory.) The alternative placement avoids this extra mechanism by handling both DMA and processor requests with the same mechanism. However, there is more interference between the processor and I/O activity.

Increased memory chip density and the cache performance tradeoff resulted in a significant component reduction. The 11/70 cache occupies four printed circuit boards (approximately 440 chips); the 11/60 occupies less than one board (approximately 85 chips). This factor-of-five component reduction is due to: (1) absence of the 32-bit bus, (2) simpler cache organization, and (3) semiconductor technology advances; these three factors contributed in approximately equal proportions.

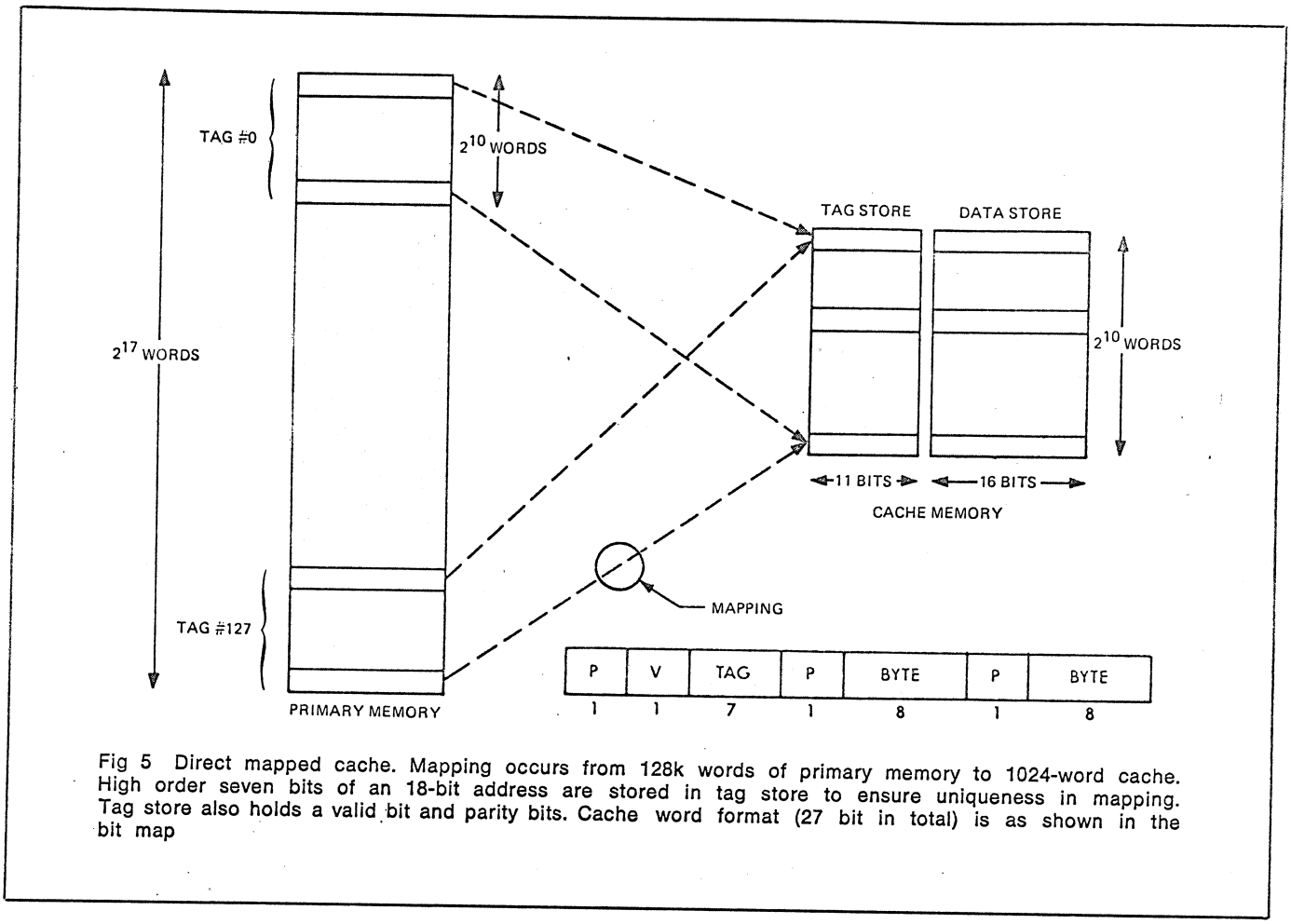


Fig 5 Direct mapped cache. Mapping occurs from 128k words of primary memory to 1024-word cache. High order seven bits of an 18-bit address are stored in tag store to ensure uniqueness in mapping. Tag store also holds a valid bit and parity bits. Cache word format (27 bit in total) is as shown in the bit map

Frequency Driven Design

Because the 11/60 implemented a stable, mature instruction set, several years of programming experience were incorporated into the system design. A simulator program was used to gather execution statistics on a range of programs. Frequency distributions of operation codes and addressing modes drove the design of the base 11/60 and the floating-point processor option.

Functions implemented in hardware, as opposed to microcode, require less time to execute. However, microprogrammed implementations are less expensive, as shown in Fig 3. Frequency distributions of operation codes guided the tradeoff. A balanced mixture of hardwired and microprogrammed implementation of functions produced a central processor that approached the speed of a computer with completely hardwired control functions, but at a lower cost.

Frequency distributions of floating-point operands were also used. Sweeney[6] analyzed the execution of more than one-million floating-point additions and tabulated the behavior of preshift alignment and postshift normalization. Both distributions are highly skewed toward low numbers of shifts. By exploiting these data, the floating-point processor performs a double-precision add in 1.02 μs as compared with 1.68 μs on a comparable unit that uses a conventional algorithm.

To measure the price/performance advantage claimed for the frequency-driven design approach in the base 11/60, a similar machine was needed for comparison. Obviously, such a machine, realized in the same semiconductor technology and designed so that the hardware resources were divided equally among all instructions, was not available. However, data were available on floating-point implementations. The floating-point processor design was a four printed circuit board unit which exploited the frequency distributions of operation codes, addressing modes, and shift amounts. A theoretical comparison was made with another 4-board design which did not use a frequency-driven approach. The 11/60 floating-point processor was estimated to exhibit a performance gain of 30 to 40% on the standard set of benchmark programs used throughout the design process.

Integral Floating-Point Arithmetic Unit

Addition of an integral floating-point arithmetic unit to the 11/60 was a direct consequence of market feedback. In particular, it was determined that the majority of the machine's users would use FORTRAN IV as a source language. In addition, among those using that language, many were not interested in heavy floating-point computation because integer arithmetic dominated their applications.

The FORTRAN IV-Plus compiler has been optimized for execution speed (as opposed to compile speed)—typically a factor of three over other available FORTRAN IV compilers. This compiler, however, employs the instruction set and auxiliary registers of the PDP-11 floating-point

processors. Thus, to take advantage of the compiler's efficiency without burdening the user with the cost of a fast floating-point processor, the central processor must provide those floating-point instructions. This is done by emulating the 46 instructions, including the 64-bit data operations, of the full floating-point instruction set using the 16-bit wide data path of the base 11/60. For users who require FORTRAN IV, but have low floating-point content in their programs, the integral floating-point unit is all that is necessary.

Additional microcode and register space added a few percent to the CPU cost. However, for that small cost increase, FORTRAN IV performance on integer programs was increased by 300%—a dramatic increase.

Cabinet-Level Integration

Physical packaging of minicomputer systems involves another set of tradeoffs. Several levels of size integration are available, ranging from the chip level (LSI-11), through the board level (11/04) and the box level (11/34), to the cabinet level (11/60).

At the cabinet level, packaging techniques are generally traditional. System fabrication is frequently the result of determining methods to install subassemblies into standard racks. At this configurator level, generalized subassemblies are usually chosen for certain functions.

This generally evokes a cost. For instance, there may be a great deal of unused space in conventional industrial racks; in most cases this excess space is simply covered with blank paneling. The cooling system, however, must be designed as if all the racks within the cabinet were occupied with subassemblies.

It was projected that the majority of the configurations sold would be system oriented; thus, design optimization at the cabinet level would be worthwhile. Therefore, the standard 11/60 is cabinet packaged. Fig 6 shows how the CPU, memory, disc units, power supplies, and expansion backplane are packaged to gain the advantages that stem from cabinet-level integration. This integration also yielded added volume, allowing a more powerful blower system to be installed. Acoustic sound power emittance is very low, considering that the rated operating environment is DEC Standard 102 Class C (122°F) for the processor. Improved power efficiency, appearance for the office environment, and subassembly accessibility are also provided.

User Microprogramming Option

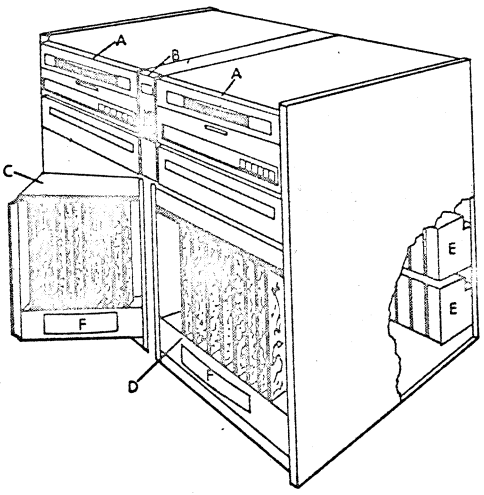
User microprogramming was incorporated in the system to meet growing market demands. The option allows users to create instructions that tailor the central processor, particularly the data flow, to his particular application.

Many potential applications of microprogramming were considered during the design of the data path and control sections of the 11/60. They ranged from instruction set extensions, eg, translation, string, and decimal arithmetic operations, to application kernels, such as node manipulation in list processing and fast Fourier transform in signal processing. Merely substituting RAM for ROM control does not result in a microprogrammable computer. A microprogrammable computer system should have the following:

(a) Extra address space in the control store
(b) Generality in the data path's processing elements
(c) A means to load the writable control store (WCS)
(d) User-oriented hardware documentation
(e) Software to support writing and debugging microprograms
(f) Integration of hardware and software protocols

All these capabilities were designed into the 11/60 WCS option.

A previously reserved operation code, 0767XX in the PDP-11 instruction set, has been allocated for users. Its designation is XFC, extended function code. When this code is recognized, the CPU transfers control to the upper 1024-word block of the 4096-word microprogram address space. User-written microcode may take over from there.



LEGEND:
A-DISC DRIVES
B-MAINTENANCE CONSOLE
C-CARD CAGE SWUNG INTO MAINTENANCE-ACCESS POSITION
D-CARD CAGE IN CLOSED POSITION
E-REAR ACCESS MODULAR POWER SUPPLIES
F-BLOWER SYSTEM

Fig 6 Cabinet packaging. Primary design goals were reliability and maintainability. System logic is mounted on swing-out card cages (C and D) for easy access. Rear access power supplies (E) are modular. Cable routing reduces electrical noise and crosstalk. Blower system (F) keeps all devices cool. Keypad (B) with numerical display facilitates machine control and maintenance. Discs (A) are top or front loading units

A second (asynchronous) type of entry to user's microcode is also provided. This occurs when a WCS-serviced interrupt is recognized by the base machine. Thus, a user can write interrupt service routines in microcode and invoke them without the usual interrupt overhead. Such routines may even be complete I/O channel emulations.

Implementation of the basic 11/60 demonstrated flexibility of microprogramming. The techniques were used in such diverse functions as console service, error logging, floating-point arithmetic, and cache initialization.

Microprogramming does not always result in significant performance gains. Well-suited applications can gain by a factor of five; poorly suited ones may give only minimal improvement. This is supported by measurements on digital signal processing software reported by Morris and Mudge[7]. Prospective users must carefully analyze the execution behavior of the application to determine which parts are "hot spots," ie, most frequently executed. For the average application, an overall factor-of-two improvement should be expected. This average, found to be a useful rule of thumb, is derived by assuming that all hot spots are microprogrammed and the remainder of the program is left unchanged.

Two user-microprogramming options are available. The first is composed of the writable control store module, software tools, and associated manuals. The second is a board containing control logic and sockets ready for the insertion of custom programmable-ROMs (p/ROMs) containing microprograms developed with the writable control store. This extended control store (ECS) option is designed for situations where microcode integrity and/or multiple installations are required.

A novel structuring of the writable control store allows it to be used to store data. Availability of data storage local to a processor, ie, not accessed through a main, general-purpose memory bus, can increase system speed. Such local store is usually implemented in some special technology that has low capacity but high performance. Writable control store has been structured so that the 48-bit microinstruction storage words can be read and written as 16-bit data words. In addition to conventional writable control store hardware, logic is available to realize a local store address register (LSAR) and a local store data register (LSDR).

Thus, the microprogrammer has fast local store available. This storage is block-oriented. A 3-cycle overhead is needed to start a block read (or block write); then, words are read (or written) at the rate of one per microcycle. The microprogram can be logically partitioned into two sections: control store—48-bit control words; and local store—16-bit data words (three per microword). A common partitioning would be 512 words of control store and 1536 words of local store.

Reliability and Maintainability

Design decisions to allocate a portion of the cost of the 11/60 to reliability and maintainability, rather than to further improving performance, were motivated by user and market needs. Prime considerations were the increasing labor cost associated with maintenance and the growing use of minicomputers in applications demanding more reliability.

First goal was to increase the mean time between failures (MTBF) by (a) reducing the occurrence and impact of normally fatal hardware malfunctions, (b) providing error statistics, and (c) providing operating alternatives to keep the system running after failures occur, albeit at a lower performance.

Second goal was to reduce the mean time to repair (MTTR) when hardware malfunctions occur, by (a) hardware design and packaging that facilitate error diagnosis and repair during scheduled and nonscheduled maintenance, (b) continuous logging of hardware errors during system operation, and (c) provision of software and microdiagnostic tools for problem isolation.

MTBF

Reducing the incidence of fatal hardware malfunctions was a joint effort by engineering and manufacturing. The Schottky transistor-transistor logic (TTL) used in the machine, having been in widespread use for over five years, is a well-proven family of devices. Moreover, conservative electrical design practices were followed.

Plotted against time, chip failure rate tends to follow a bathtub-shaped curve, high at either end of the life cycle. The 11/60 production process includes extensive thermal cycling to ensure that "infant mortality" cases are discovered early during manufacturing.

The cabinet is designed to minimize buildup of hot air over the processor boards. Power supplies are mounted at the rear of the cabinet, away from the logic, so that radiant heating effects are minimized. A blower system cools the logic card cage by drawing fresh, filtered air down over the printed circuit boards such that no board receives exhaust air from another.

Other physical packaging to reduce hardware problems include cable troughs, impact-absorbing casters, and special cabinet grounding. A filter is attached to the maintenance console to reduce electrostatic noise interference.

Console microcode double checks every entry to verify data received from the keypad. A significant proportion of the 11/60 microcode (see Table) is devoted to logging microlevel state upon the occurrence of a detected error. This logged state can be accessed via a maintenance examine and deposit (MED) instruction. Logged information is used by an operating system to compile error records, which aid in tracking down intermittent errors.

To reduce the impact of hardware malfunctions on the user environment, a number of fail-soft capabilities have been implemented.

(1) If the cache fails, it is turned off and the still functioning primary memory is used to keep the system running.

(2) If a parity error occurs in WCS, the processor disables that control store. Then, the operating system is notified, and program execution can continue using the basic PDP-11 instructions.

(3) Systems can be programmed to fall back onto the integral floating-point unit if an error is detected in the floating-point processor.

(4) The bootstrap loader permits system loading from an alternative device if the primary bootstrapping device is disabled.

Control Store Usage by Category

Category		Number of Microwords	Percentage of Total
A	PDP-11 Instruction Set		
	Initialization	95	4
	Operand fetch, execution, and operand store	515	20
	Infrequent intraprocessor transfers	230	9
		840	
B	Integral Floating-Point Instruction Set	1010	40
C	Reliability and Maintainability		
	Error logging, MED, and cache fail-soft	190	7
	Console, boot, and initial diagnostic	230	9
D	Support of Options		
	Writable control store	60	2
	Floating-point processor	80	3
E	Reserved for Future Changes and Additions	150	6
		2560	100

Total address space for microprograms is 4096 words, of which the 2560 categorized in the table are implemented in ROM. Note also the increased utilization of microprogramming in the 11/60, as compared to the 11/40.

Category A, totaling 840 words, was implemented in 256 words for the 11/40. The two machines have comparable microword widths. The third subcategory in Category A illustrates the use of microprogramming in the frequency-driven design approach. Examples of infrequent intraprocessor transfers are error handling and data transfer to and from internal addresses, eg, memory management relocation registers.

One of the benefits of a microprogrammed implementation of control is the ease with which engineering change orders (ECO) can be implemented. Space in Category E is reserved for such use and for the further correction of undetected errors in the microcode itself.

MTTR

Error diagnosis is the most time-consuming problem facing the field-service engineer. Special diagnostic tools, both hardware and software, have been designed to reduce the time spent in error isolation.

Focal point of the hardware maintainability effort is the microdiagnostic unit. This single board tests the logic on five of the six processor boards. When faults are detected, an error code is displayed on light-emitting diodes (LEDs). A fault directory can then be used to determine which boards are to be replaced. The unit requires only a small portion of the internal machine (the microword sequencing) to be operational.

In addition, a number of onboard diagnostic aids are included in the CPU design. These include LEDs to display the contents of the next microaddress register, a single-step mode, and a microbreak function.

Software diagnostic programs are used to diagnose errors in system peripherals as well as in all CPU subsystems, such as memory management unit and cache. User-mode diagnostic programs allow peripheral diagnosis to occur while the system is available for other users. Conventional standalone diagnostic programs can also be used.

Physical packaging facilitates quick repair. Hinged card cages and modular power supplies allow easy access and module change.

Summary

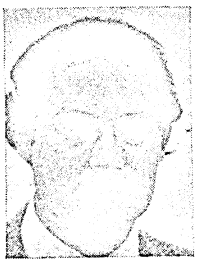
The design of a mid-range minicomputer has been used as a concrete illustration of tradeoffs made to effect a price/performance balance. Designers use technology advances, eg, doubling of density on a memory chip, to produce new designs in one of two design styles: constant-cost, increasing-functionality or constant-functionality, decreasing-cost. Increased use of microprogramming, a factor-of-three in this case study, is a trend that was observed.

By choosing a less powerful cache organization, the 11/60 design obtained a factor-of-five component reduction. Cache design also illustrates how some design parameters are highly interdependent. The frequency driven design approach used on the floating-point processor can lead to a 40% performance gain.

Examples of added functionality in the constant-cost style of design include greater reliability and maintainability, and user microprogramming.

References

1. G. Bell, *et al*, "A New Architecture for Minicomputers—The DEC PDP-11," *Proceedings of the 1970 Spring Joint Computer Conference*, pp 657-675
2. W. H. Davidow, "The Rationale for Logic from Semiconductor Memory," *Proceedings of the 1972 Spring Joint Computer Conference*, pp 353-358
3. J. F. O'Loughlin, "Microprogramming a Fixed Architecture Machine, Microprogramming and Systems Architecture," *Infotech State of the Art Report 23*, 1975, pp 205-224
4. W. D. Strecker, "Cache Memories for PDP-11 Family Computers," *Proceedings of the 3rd Annual Symposium on Computer Architecture*, 1976, pp 155-158
5. R. M. Meade, "Design Approaches for Cache Memory Control," *Computer Design*, Jan 1971, pp 87-93
6. D. W. Sweeney, "An Analysis of Floating-Point Addition," *IBM Systems Journal*, 1965, pp 31-42
7. L. R. Morris and J. C. Mudge, "Speed Enhancement of Digital Signal Processing Software Via Microprogramming a General Purpose Minicomputer," *Conference Record*, IEEE International Conference on Acoustics, Speech, and Signal Processing, May 1977



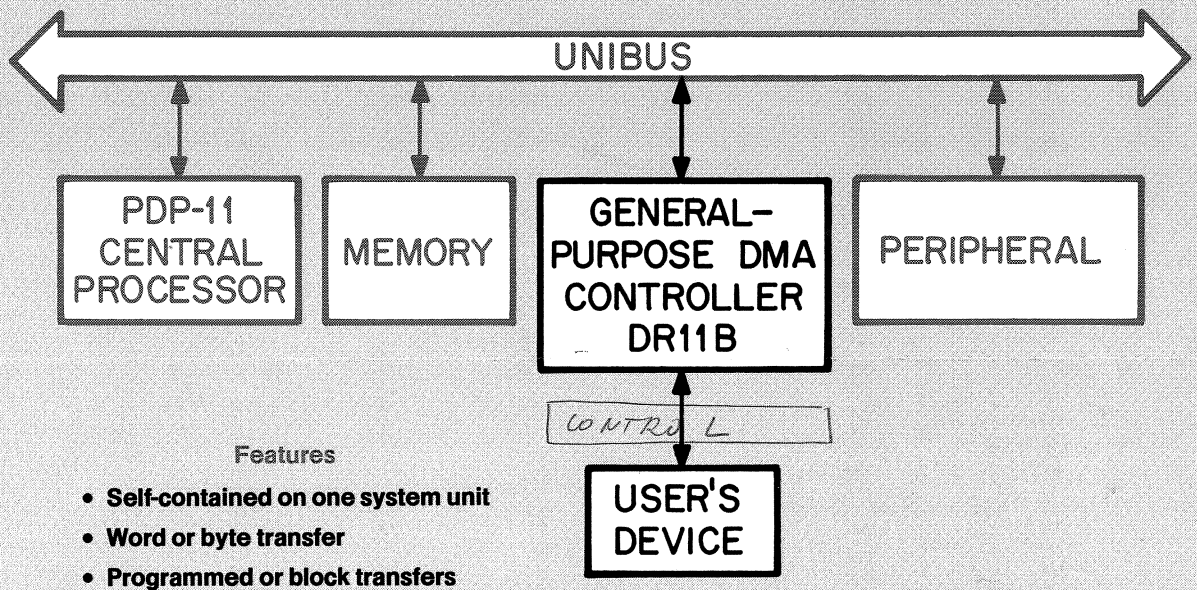
Craig Mudge is a consulting engineer in PDP-11 engineering at Digital Equipment Corp. He holds a PhD degree in computer science from the University of North Carolina at Chapel Hill and a Bachelor's degree from the Australian National University, Canberra. His background includes CPU development and applications programming, as well as systems architecture.

DIGITAL EQUIPMENT CORPORATION

OPTION BULLETIN

FEBRUARY 1973

Direct Memory Access Controller (DR11-B)



Features

- **Self-contained on one system unit**
- **Word or byte transfer**
- **Programmed or block transfers**
- **Processor-to-processor data transfer**
- **Built-in maintenance feature**
- **User-controlled transfer rates up to memory speed**

GENERAL DESCRIPTION

The DR11-B is a general-purpose, direct-memory access controller for the PDP-11 UNIBUS™.

The controller is self-contained, plugs into the UNIBUS, and allows the user to control data transfers between a PDP-11 system and his device. The controller is similar to the DR11-A in function, but rather than using programmed-controlled transfers, it operates via direct-memory access at rates defined by the user.

The controller has four registers: command and status, word count, bus address, and data. Operation is initiated under program control by loading the word count register with the 2's complement of the number of transfers, specifying the initial memory or bus address where the block transfer is to begin, and by loading the command/status register with function bits. The user's device recognizes these function bits and responds by setting up the control inputs. The user's device requests the bus and moves data to or from memory at a user-defined rate until the specified number of words are transferred.

The user has a number of control lines for flexibility. For example, burst modes, read-modify restore operations, and byte transfers are possible within the control structure.

PHYSICAL DESCRIPTION

The DR11-B is contained on one standard system unit for convenient incorporation into a PDP-11 system. A UNIBUS jumper module (M920) is supplied with the unit. Power is applied to the logic through a power harness already provided in the mounting box. Power requirements are 3.3 amps at + 5v.

Connections to the user's device are made through two, split-lug, cable boards supplied with the unit. Alternatively, an M920 UNIBUS connector module can be used to jumper all user signals to an adjacent BB11 blank mounting panel, which can be used to package some or all of the device logic (neither the additional M920 nor the BB11 is supplied with the unit).

STATUS AND COMMAND REGISTER (DRST)

The DRST is used to give commands to the user's device and also provides status indicators of the DR11-B control and the user's device to the program.

WORD COUNT REGISTER (DRWC)

DRWC is a 16-bit, read/write register. It is initially loaded with the two's complement of the number of transfers to be made, and normally increments up towards zero after each bus cycle. Incrementation can be inhibited by the user device. When overflow occurs (all 1's to all 0's), an interrupt request is generated, and bus cycles stop.

BUS ADDRESS REGISTER (DRBA)

DRBA is a 15-bit, read/write register; bit 0, corresponding to address line A00, is not implemented, but can be provided by the user's device. Two additional bits in the status and command register are used to specify BUS A (17:16) in direct bus transfers. The register is normally incremented (although the user can inhibit) after each bus cycle, advancing the address to the next sequential word location on the bus.

DATA BUFFER REGISTER (DRDB)

The DRDB serves two functions: First, it is a 16-bit, write-only register. The outputs of this register are available to the user's device. This register can be loaded under program control, and can also be used to buffer information when data is being transferred from the UNIBUS to the user device (in block operation).

Second, the DRDB functions as a 16-bit, read-only register. Information to be read is provided by the user's device on the DATA IN signal lines. Information may be read under program control, or automatically, for block operations.

MAINTENANCE MODE

Checkout and test of the DR11-B is accomplished by using a MAINT bit in DRST along with a special maintenance module which simulates the user's device. The maintenance module plugs directly into the two slots normally occupied by the cable boards and jumps the output and input signals. The maintenance module is included with the DR11-B.

Register Assignments	Addresses	Vector
1st DR11-B	772410-772417	124
2nd DR11-B	772430-772437	*
3rd DR11-B	772450-772457	*
4th DR11-B	772470-772477	*

*assigned by user

Priority Level

Bus Request 5

Input/Output Levels (standard TTL)

+3V=1; OV=0

Environmental

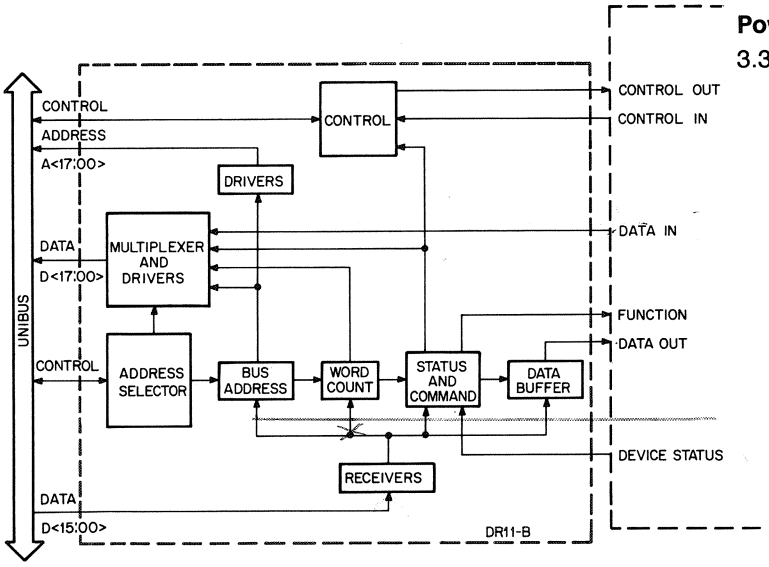
Temperature	10° -50° C
Humidity	20%-90% non-condensating

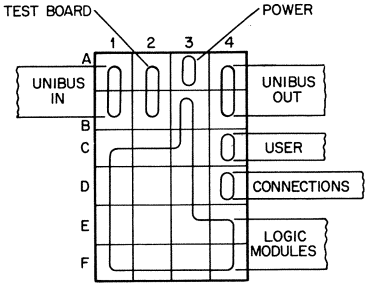
Physical

Contained on one system unit; can be mounted in processor or expander box. Power delivered from H720 in BA11.

Power Requirements

3.3 amps at + 5V





digital

146 Main Street, Maynard, Massachusetts, U.S.A. 01754

Reprinted from –
AFIPS – Conference Proceedings, Volume 36

Montvale, New Jersey
07645

A new architecture for mini-computers— The DEC PDP-11

by G. BELL,* R. CADY, H. McFARLAND, B. DELAGI, J. O'LAUGHLIN and R. NOONAN

Digital Equipment Corporation
Maynard, Massachusetts

and

W. WULF

Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

The mini-computer** has a wide variety of uses: communications controller; instrument controller; large-system pre-processor; real-time data acquisition systems . . .; desk calculator. Historically, Digital Equipment Corporation's PDP-8 Family, with 6,000 installations has been the archetype of these mini-computers.

In some applications current mini-computers have limitations. These limitations show up when the scope of their initial task is increased (e.g., using a higher level language, or processing more variables). Increasing the scope of the task generally requires the use of more comprehensive executives and system control programs, hence larger memories and more processing. This larger system tends to be at the limit of current mini-computer capability, thus the user receives diminishing returns with respect to memory, speed efficiency and program development time. This limitation is not surprising since the basic architectural concepts for current mini-computers were formed in the early 1960's. First, the design was constrained by cost, resulting in rather simple processor logic and register configurations. Second, application experience was not available. For example, the early constraints often created computing designs with what we now consider weaknesses:

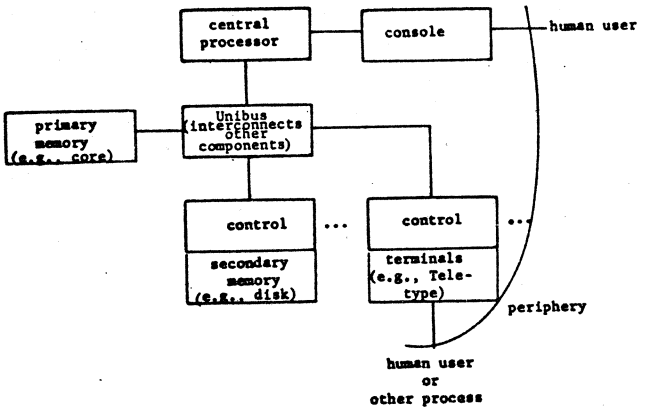
1. limited addressing capability, particularly of larger core sizes
2. few registers, general registers, accumulators, index registers, base registers
3. no hardware stack facilities
4. limited priority interrupt structures, and thus slow context switching among multiple programs (tasks)
5. no byte string handling
6. no read only memory facilities
7. very elementary I/O processing

---

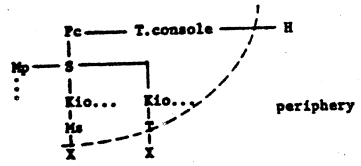
* Also at Carnegie–Mellon University, Pittsburgh, Pennsylvania.

** The PDP-11 design is predicated on being a member of one (or more) of the micro, midi, mini, . . ., maxi (computer name) markets. We will define these names as belonging to computers of the third generation (integrated circuit to medium scale integrated circuit technology), having a core memory with cycle time of .5 ~ 2 microseconds, a clock rate of 5 ~ 10 *Mhz* . . ., a single processor with interrupts and usually applied to doing a particular task (e.g., controlling a memory or communications lines, pre-processing for a larger system, process control). The specialized names are defined as follows:

	maximum addressable primary memory (words)	*processor and memory cost (1970 kilodollars)*	*word length (bits)*	*processor state (words)*	*data types*
micro	8 K	~ 5	8 ~ 12	2	integers, words, boolean vectors
mini	32 K	5 ~ 10	12 ~ 16	2–4	vectors (i.e., indexing)
midi	65 ~ 128 K	10 ~ 20	16 ~ 24	4–16	double length floating point (occasionally)



Conventional block diagram



PMS diagram[1]

[1] PMS Notation

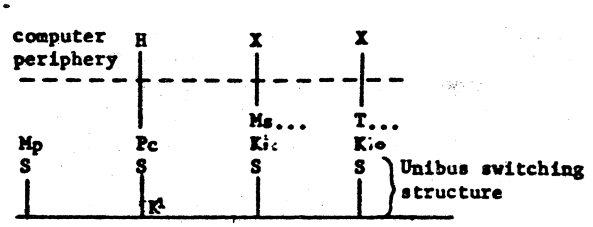
form	comment
Component/X	name X is an alias (abbreviation for a component is separated by /)
a := b	a is assigned the meaning of b
|	delimits mutually exclusive alternatives
Components := (Processor/P | Memory/M|Switch/S| Control/K|Terminal/T| Data operation/D|Link/L| Human/H)	set of primitive components and their abbreviations
$X(a_1:v_1; a_2:v_2; \ldots a_n:v_b)$	a attribute/a, value/v pairs. Attribute may be omitted if it can be inferred from dimensions of value.
index number/#	attribute giving component number
name/'	attribute giving component name
miscellaneous abbreviations := (Mp/primary memory|Ms/secondary memory|Pc/central processor| Kio/io control|Pio/io processor|s/sec/seconds|char/character| b/bit|w/word|i/information|	
———	information carrying link (bi-directional)
→←	uni-directional information carrying links
	delimits alternatives

Figure 1—Conventional block diagram and PMS diagram of PDP-11

use the bus for a message (call). The S (Unibus)differs from most switches because any component can communicate with any other component.

The types of messages in the PDP-11 are along the lines of the hierarchical structure common to present day computers. The single bus makes conventional and other structures possible. The message processes in the structure which utilize S(Unibus) are:

1. The central processor (Pc) requests that data be read or written from or to primary memory (Mp) for instructions and data. The processor calls a particular memory module by concurrently specifying the module's address, and the address within the modules. Depending on whether the processor requests reading or writing, data is transmitted either from the memory to the processor or vice versa.
2. The central processor (Pc) controls the initialization of secondary memory (Ms) and terminal (T) activity. The processor sets status bits in the control associated with a particular Ms or T, and the device proceeds with the specified action (e.g., reading a card, or punching a character into paper tape). Since some devices transfer data vectors directly to primary memory, the vector control information (i.e., the memory location and length) is given as initialization information.
3. Controls request the processor's attention in the form of interrupts. An interrupt request to the processor has the effect of changing the state of the processor; thus the processor begins executing a program associated with the interrupting process. Note, the interrupt process is only a signaling method, and when the processor interruption occurs, the interruptee specifies a unique address value to the processor. The address is a starting address for a program.
4. The central processor can control the transmission of data between a control (for T or Ms) and either the processor or a primary memory for program controlled data transfers. The device signals for attention using the interrupt dialogue and the central processor responds by managing the data transmission in a fashion similar to transmitting initialization information.



[1] Unibus control packaged with Pc

Figure 2—PDP-11 physical structure PMS diagram

unnecessary because each device interrupt corresponds to a unique address.

Software

The total system including software is of course the main objective of the design. Two techniques were used to aid programmability: first benchmarks gave a continuous indication as to how well the machine interpreted programs; second, systems programmer continually evaluated the design. Their evaluation considered: what code the compiler would produce; how would the loader work; ease of program relocability; the use of a debugging program; how the compiler, assembler and editor would be coded—in effect, other benchmarks; how real time monitors would be written to use the various facilities and present a clean interface to the users; finally the ease of coding a program.

Modularity

Structural flexibility (sometimes called modularity) for a particular model was desired. A flexible and straightforward method for interconnecting components had to be used because of varying user needs (among user classes and over time). Users should have the ability to configure an optimum system based on cost, performance and reliability, both by interconnection and, when necessary, constructing new components. Since users build special hardware, a computer should be easily interfaced. As a by-product of modularity, computer components can be produced and stocked, rather than tailor-made on order. The physical structure is almost identical to the PMS structure discussed in the following section; thus, reasonably large building blocks are available to the user.

Microprogramming

A note on microprogramming is in order because of current interest in the "firmware" concept. We believe microprogramming, as we understand it (Wilkes, 1951), can be a worthwhile technique as it applies to processor design. For example, microprogramming can probably be used in larger computers when floating point data operators are needed. The IBM System/360 has made use of the technique for defining processors that interpret both the System/360 instruction set and earlier family instruction sets (e.g., 1401, 1620, 7090). In the PDP-11 the basic instruction set is quite straightforward and does not necessitate microprogrammed interpretation. The processor-memory connection is asynchronous and therefore memory of any speed can be connected. The instruction set encourages the user to write reentrant programs; thus, read-only memory can be used as part of primary memory to gain the permanency and performance normally attributed to microprogramming. In fact, the Model 10 computer which will not be further discussed has a 1024-word read only memory, and a 128-word read-write memory.

Understandability

Understandability was perhaps the most fundamental constraint (or goal) although it is now somewhat less important to have a machine that can be quickly understood by a novice computer user than it was a few years ago. DEC's early success has been predicated on selling to an intelligent but inexperienced user. Understandability, though hard to measure, is an important goal because all (potential) users must understand the computer. A straightforward design should simplify the systems programming task; in the case of a compiler, it should make translation (particularly code generation) easier.

PDP-11 STRUCTURE AT THE PMS LEVEL*

Introduction

PDP-11 has the same organizational structure as nearly all present day computers (Figure 1). The primitive PMS components are: the primary memory (Mp) which holds the programs while the central processor (Pc) interprets them; io controls (Kio) which manage data transfers between terminals (T) or secondary memories (Ms) to primary memory (Mp); the components outside the computer at periphery (X) either humans (H) or some external process (e.g., another computer); the processor console (T. console) by which humans communicate with the computer and observe its behavior and affect changes in its state; and a switch (S) with its control (K) which allows all the other components to communicate with one another. In the case of PDP-11, the central logical switch structure is implemented using a bus or chained switch (S) called the Unibus, as shown in Figure 2. Each physical component has a switch for placing messages on the bus or taking messages off the bus. The central control decides the next component to

* A descriptive (block-diagram) level (Bell and Newell, 1970) to describe the relationship of the computer components: processors memories, switches, controls, links, terminals and data operators.

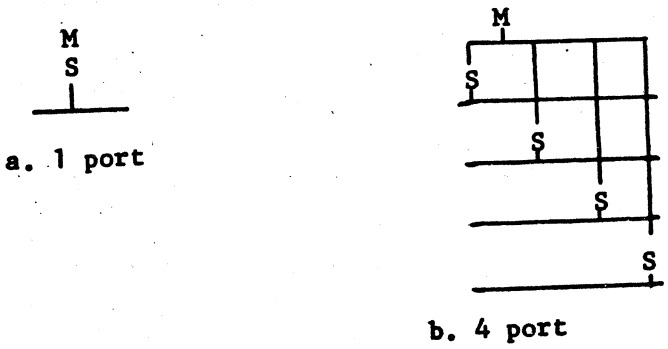


Figure 6—1 and 4 port memory modules PMS diagram

transferring data to multiple primary memories. On a larger system with multiple independent memories the supply of memory cycles is 17 megabits/second times the number of modules. Since there is such a large supply of memory cycles/second and since the central processor can only absorb approximately 16 megabits/second, the simple one Unibus structure must be modified to make the memory cycles available. Two changes are necessary: first, each of the memory modules have to be changed so that multiple units can access each module on an independent basis; and second, there must be independent control accessing mechanisms. Figure 6 shows how a single memory is modified to have more access ports (i.e., connect to 4 Unibusses).

Figure 7 shows a system with 3 independent memory modules which are accessed by 2 independent Unibusses. Note that two of the secondary memories and one of the transducers are connected to both Unibusses. It should be noted that devices which can potentially interfere with Pc-Mp accesses are constructed with two ports; for simple systems, the two ports are both connected to the same bus, but for systems with more busses, the second connection is to an independent bus.

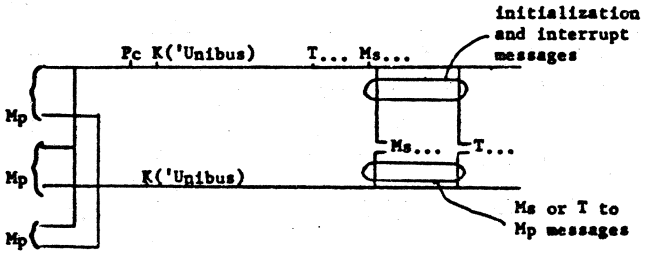


Figure 7—Three Mp, 2 S('Unibus) structure PMS diagram

Figure 8 shows a multiprocessor system with two central processors and three Unibusses. Two of the Unibus controls are included within the two processors, and the third bus is controlled by an independent control unit. The structure also has a second switch to allow either of two processors (Unibusses) to access common shared devices. The interrupt mechanism allows either processor to respond to an interrupt and similarly either processor may issue initialization information on an anonymous basis. A control unit is needed so that two processors can communicate with one another; shared primary memory is normally used to carry the body of the message. A control connected to two Pc's (see Figure 8) can be used for reliability; either processor or Unibus could fail, and the shared Ms would still be accessible.

Higher performance processors

Increasing the bus width has the greatest effect on performance. A single bus limits data transmission to 21.4 megabits/second, and though Model 20 memories are 16 megabits/second, faster (or wider) data path width modules will be limited by the bus. The Model 20 is not restricted, but for higher performance processors operating on double word (fixed point) or triple word (floating point) data two or three accesses are required for a single data type. The direct method to improve the performance is to double or triple the primary memory and central processor data path widths. Thus, the bus data rate is automatically doubled or tripled.

For 32- or 48-bit memories a coupling control unit is needed so that devices of either width appear isomorphic to one another. The coupler maps a data

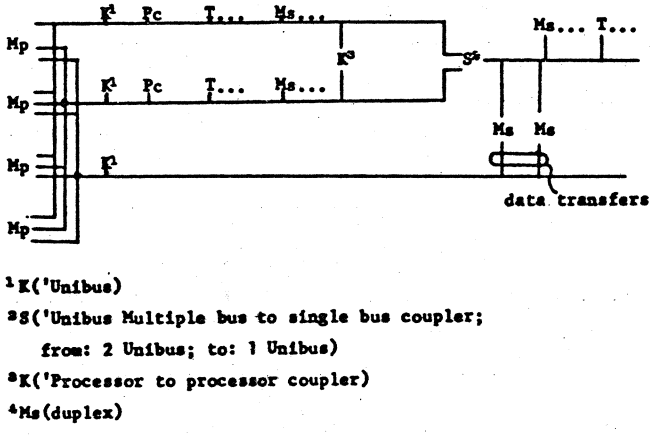


Figure 8—Dual Pc multiprocessor system PMS diagram

5. Some device controls (for T or Ms) transfer data directly to/from primary memory without central processor intervention. In this mode the device behaves similar to a processor; a memory address is specified, and the data is transmitted between the device and primary memory.
6. The transfer of data between two controls, e.g., a secondary memory (disk) and say a terminal/T. display is not precluded, provided the two use compatible message formats.

As we show more detail in the structure there are, of course, more messages (and more simultaneous activity). The above does not describe the shared control and its associated switching which is typical of a magnetic tape and magnetic disk secondary memory systems. A control for a DECtape memory (Figure 3) has an S('DECtape bus) for transmitting data between

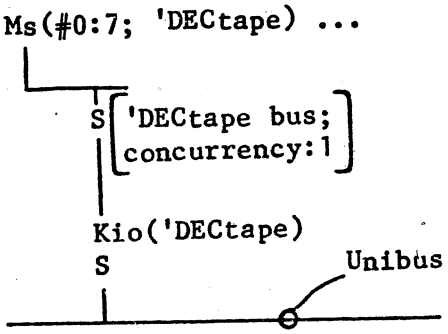


Figure 3—DECtape control switching PMS diagram

a single tape unit and the DECtape transport. The existence of this kind of structure is based on the relatively high cost of the control relative to the cost of the tape and the value of being able to run concurrently with other tapes. There is also a dialogue at the periphery between X-T and X-Ms which does not use the Unibus. (For example, the removal of a magnetic tape reel from a tape unit or a human user (H) striking a typewriter key are typical dialogues.)

All of these dialogues lead to the hierarchy of present computers (Fig. 4). In this hierarchy we can see the paths by which the above messages are passed (Pc-Mp; Pc-K; K-Pc; Kio-T and Kio-Ms; and Kio-Mp; and, at the periphery, T-X and T-Ms; and T.console-H).

Model 20 implementation

Figure 5 shows the detailed structure of a uniprocessor, Model 20 PDP-11 with its various

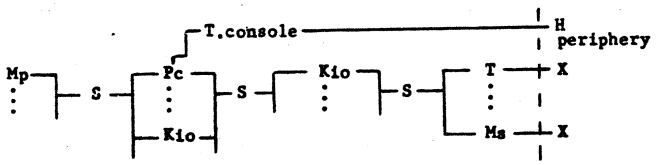


Figure 4—Conventional hierarchy computer structure

components (options). In Figure 5 the Unibus characteristics are surpressed. (The detailed properties of the switch are described in the logical design section.)

Extensions to increase performance

The reader should note (Figure 5) that the important limitations of the bus are: a concurrency of one, namely, only one dialogue can occur at a given time, and a maximum transfer rate of one 16-bit word per .75 μsec., giving a transfer rate of 21.3 megabits/second. While the bus is not a limit for a uni-processor structure, it is a limit for multiprocessor structures. The bus also imposes an artificial limit on the system performance when high speed devices (e.g., TV cameras, disks) are

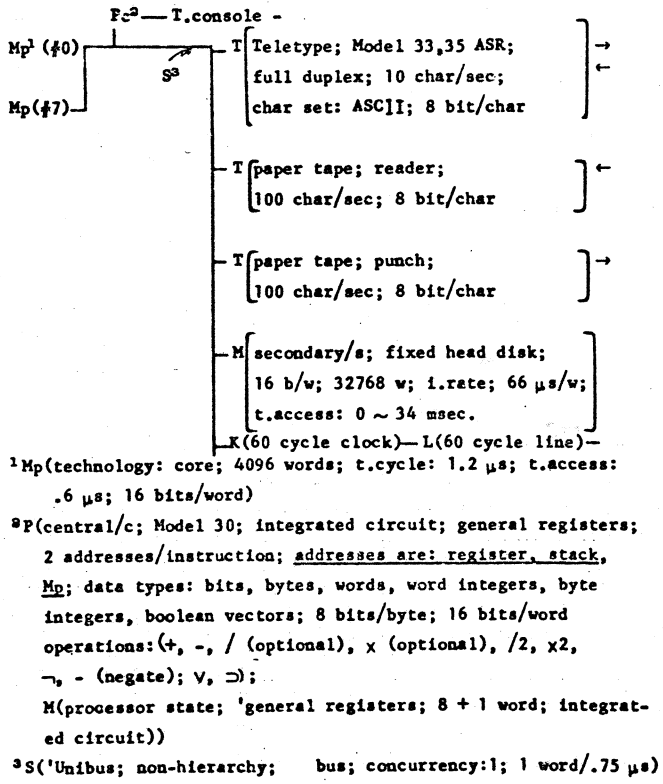


Figure 5—PDP-11 structure and characteristics PMS diagram

access method for specifying algorithms, since very little space, only the access addresses and the operators, needs to be given. In this scheme the operands of an operator are always assumed to be on the "top of the stack". The stack has the additional advantage that arithmetic expression evaluation and compiler statement parsing have been developed to use a stack effectively. The disadvantage of the stack is due in part to the nature of current memory technology. That is, stack memories have to be simulated with random access memories, multiple stacks are usually required, and even though small stack memories exist, as the stack overflows, the primary memory (core) has to be used.

Even though the trend has been toward the general register concept (which, of course, is similar to a two address scheme in which one of the addresses is limited to small values), it is important to recognize that any design is a compromise. There are situations for which any of these schemes can be shown to be "best". The IBM System/360 series uses a general register structure, and their designers (Amdahl, Blaauw and Brooks, 1964) claim the following advantages for the scheme:

1. Registers can be assigned to various functions: base addressing, address calculation, fixed point arithmetic and indexing.
2. Availability of technology makes the general registers structure attractive.

The System/360 designers also claim that a stack organized machine such as the English Electric KDF 9 (Allmark and Lucking, 1962) or the Burroughs B5000 (Lonegran and King, 1961) has the following disadvantages:

1. Performance is derived from fast registers, not the way they are used.
2. Stack organization is too limiting and requires many copy and swap operations.
3. The overall storage of general registers and stack machines are the same, considering point #2.
4. The stack has a bottom, and when placed in slower memory there is a performance loss.
5. Subroutine transparency is not easily realized with one stack.
6. Variable length data is awkward with a stack.

We generally concur with points 1, 2, and 4. Point 5 is an erroneous conclusion, and point 6 is irrelevant (that is, general register machines have the same problem). The general-register scheme also allows processor implementations with a high degree of parallelism since instructions of a local block all can operate on several registers concurrently. A set of truly general purpose registers should also have additional uses. For example, in the DEC PDP-10, general registers are used for address integers, indexing, floating point, boolean vectors (bits), or program flags and stack pointers. The general registers are also addressable as primary memory, and thus, short program loops can reside within them and be interpreted faster. It was observed in operation that PDP-10 stack operations were very powerful and often used ((accounting for as many as 20% of the executed instructions, in some programs, e.g., the compilers.)

The basic design decision which sets the PDP-11 apart was based on the observation that by using *truly* general registers and by suitable addressing mechanisms it was possible to consider the machine as a zero-address (stack), one-address (general register), or two-address (memory-to-memory) computer. Thus, it is possible to use whichever addressing scheme, or mixture of schemes, is most appropriate.

Another important design decision for the instruction set was to have only a few data types in the basic machine, and to have a rather complete set of operations for each data type. (Alternative designs might have more data types with few operations, or few data types with few operations.) In part, this was dictated by the machine size. The conversion between data types must be easily accomplished either automatically or with 1 or 2 instructions. The data types should also be sufficiently primitive to allow other data types to be defined by software (and by hardware in more powerful versions of the machine). The basic data type of the machine is the 16 bit integer which uses the two's complement convention for sign. This data type is also identical to an address.

PDP-11 model 20 instruction set (basic instruction set)

A formal description of the basic instruction set is given in Appendix 1 using the ISPL notation (Bell and Newell, 1970). The remainder of this section will discuss the machine in a conventional manner.

Primary memory

The primary memory (core) is addressed as either 2^{16} bytes or 2^{15} words using a 16 bit number. The linear address space is also used to access the input-output devices. The device state, data and control registers are read or written like normal memory locations.

request of a given width into a higher- or lower-width request for the bus being coupled to, as shown in Figure 9. (The bus is limited to a fixed number of devices for electrical reasons; thus, to extend the bus a bus repeating unit is needed. The bus repeating control unit is almost identical to the bus coupler.) A computer with a 48-bit primary memory and processor and 16-bit secondary memory and terminals (transducers) is shown in Figure 9.

In summary, the design goal was to have a modular structure providing the final user with freedom and flexibility to match his needs. A secondary goal of the Unibus is open-endedness by providing multiple busses and defining wider path busses. Finally, and most important, the Unibus is straightforward.

THE INSTRUCTION SET PROCESSOR (ISP) LEVEL-ARCHITECTURE*

Introduction, background and design constraints

The Instruction Set Processor (ISP) is the machine defined by hardware and/or software which interprets programs. As such, an ISP is independent of technology and specific implementations.

The instruction set is one of the least understood aspects of computer design; currently it is an art. There is currently no theory of instruction sets, although there have been attempts to construct them (Maurer, 1966), and there has also been an attempt to have a computer program design an instruction set (Haney, 1968). We have used the conventional approach in this design: first a basic ISP was adopted and then incremental design modifications were made (based on the results of the benchmarks).**

Although the approach to the design was conventional, the resulting machine is not. A common classification of processors is as zero-, one-, two-, three-, or three-plus-one-address machines. This scheme has the the form:

$$op\ l1, l2, l3, l4$$

where $l1$ specifies the location (address) in which to store the result of the binary operation (op) of the contents of operand locations $l2$ and $l3$, and $l4$ specifies the location of the next instruction.

The action of the instruction is of the form:

$$l1 \leftarrow l2\ op\ l3;\ goto\ l4$$

The other addressing schemes assume specific values for one or more of these locations. Thus, the one-address von Neumann (Burks, Goldstine and von Neumann, 1946) machines assume $l1 = l2 =$ the "accumulator" and $l4$ is the location following that of the current instruction. The two-address machine assumes $l1 = l2$; $l4$ is the next address.

Historically, the trend in machine design has been to move from a 1 or 2 word accumulator structure as in the von Neumann machine towards a machine with accumulator and index register(s).* As the number of registers is increased the assignment of the registers to specific functions becomes more undesirable and inflexible; thus, the general-register concept has developed. The use of an array of general registers in the processor was apparently first used in the first-generation, vacuum-tube machine, PEGASUS (Elliott et al., 1956) and appears to be an outgrowth of both 1- and 2-address structures. (Two alternative structures—the early 2- and 3-address per instruction computers may be disregarded, since they tend to always access primary memory for results as well as temporary storage and thus are wasteful of time and memory cycles, and require a long instruction.) The stack concept (zero-address) provides the most efficient

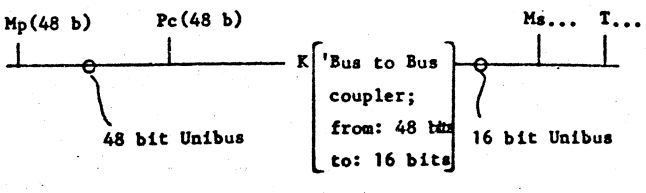


Figure 9—Computer with 48 bit Pc, Mp with 16 bit Ms, T PMS diagram

* The word architecture has been operationally defined (Amdahl, Blaauw and Brooks, 1964) as "the attributes of a system as seen by a programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design and the physical implementation."

** A predecessor multiregister computer was proposed which used a similar design process. Benchmark programs were coded on each of 10 "competitive" machines, and the object of the design was to get a machine which gave the best score on the benchmarks. This approach had several fallacies: the machine had no basic character of its own; the machine was difficult to program since the multiple registers were assigned to specific functions and had inherent idiosyncrasies to score well on the benchmarks; the machine did not perform well for programs other than those used in the benchmark test; and finally, compilers which took addvantage of the machine appeared to be difficult to write. Since all "competitive machines" had been hand-coded from a common flowchart rather than separate flowcharts for each machine, the apparent high performance may have been due to the flowchart organization.

* Due in part to needs, but mainly technology which dictates how large the structure can be.

Common stack instruction:	Equivalent PDP-11 instruction:
place address value A on stack	MOVE #A, -(R0)[1]
load stack from memory address specified by stack	MOVE @(R0)+, - (R0)
load stack from memory location A	MOVE A, -(R0)
store stack at memory address specified by stack	MOVE (R0)+, @(R0)+
store stack at memory location A	MOVE (R0)+, A
duplicate top of stack	MOVE (R0), -(R0)
+, add 2 top data of stack to stack	ADD (R0) +, @R0
-, x, /; subtract, multiply, divide	(see add)
-; negate top data of stack	NEG @R0
clear top data of stack	CLR @R0
∨; "inclusive or" 2 top data of stack "and" 2 top data of stack	BSET (R0)+, @R0
¬; complement top of stack	COM @R0
test top of stack (set branch indicators)	TST @R0
branch on indicator	BR (=, ≠, >, ≥, <, ≤)
jump unconditional	JUMP

add addressed location A to top of stack - (not common for stack machine) equivalent to: load stack, add swap top 2 stack data	ADD A, @R0
	MOVE (R0)+, R1
	MOVE (R0)+, R2
	MOVE R1, -(R0)
	MOVE R2, -(R0)

reset stack location to H	MOVE #H, R0 COM @R0
A, "and" 2 top stack data	BCLR (R0)+, @R0

[1] Stack pointer has been arbitrarily used as register R0 for this example.

Figure 14—Stack computer instructions and equivalent PDP-11 instructions

However, with the PDP-11 there is an address method for improving the program encoding and run time, while not losing the stack concept. An encoding improvement is made by doing an operation to the top of the stack from a direct memory location (while loading). Thus the previous example could be coded as:

load stack B
divide stack by C
add A to stack
store stack D

Use as a one-address (general register) machine

The PDP-11 is a general register computer and should be judged on that basis. Benchmarks have been coded to compare the PDP-11 with the larger DEC PDP-10. A 16 bit processor performs better than the DEC PDP-10 in terms of bit efficiency, but not with time or memory cycles. A PDP-11 with a 32 bit wide memory would, however, decrease time by nearly a factor of two, making the times essentially comparable.

Use as a two-address machine

Figure 15 lists typical two-address machine instructions together with the equivalent PDP-11 instructions for performing the same operations. The most useful instruction is probably the MOVE instruction because it does not use the stack or general registers. Unary instructions which operate on and test primary memory are also useful and efficient instructions.

Extensions of the instruction set for real (floating point) arithmetic

The most significant factor that affects performance is whether a machine has operators for manipulating data in a particular format. The inherent generality of a stored program computer allows any computer by subroutine to simulate another—given enough time and memory. The biggest and perhaps only factor that separates a small computer from a large computer is whether floating point data is understood by the computer. For example, a small computer with a cycle time of 1.0 microseconds and 16 bit memory width might have the following characteristics for a floating point add, excluding data accesses:

programmed:	250 microseconds
programmed (but special normalize and differencing of exponent instructions):	75 microseconds
microprogrammed hardware:	25 microseconds
hardwired:	2 microseconds

It should be noted that the ratios between programmed and hardwired interpretation varies by roughly two orders of magnitude. The basic hardwiring scheme and the programmed scheme should allow binary program compatibility, assuming there is an interpretive program for the various operators in the Model 20. For example, consider one scheme which would add eight 48 bit registers which are addressable in the extended instruction set. The eight floating registers, F, would be mapped into eight double length

Two Address Computer	PDP-11
A ← B; transfer B to A	MOVE B,A
A ← A+B; add	ADD B,A
-, x, /	(see add)
A ← -A; negate	NEG A
A ← A ∨ B; inclusive or	%SETB,A
A ← ¬ A; not	COM
Jump unconditioned	JUMP
Test A, and transfer to B	TST A BR (=, ≠, >, ≥, <, ≤) B

Figure 15—Two address computer instructions and equivalent PDP-11 instructions

General register

The general registers are named: R[0:7]⟨15:0⟩*; that is, there are 8 registers each with 16 bits. The naming is done starting (at the left with bit 15 (the sign bit) to the least significant bit 0. There are synonyms for R[6] and R[7]:

Stack Pointer/SP⟨15:0⟩ := R[6]⟨15:0⟩
: used to access a special stack which is used to store the state of interrupts, traps and subroutine calls

Program Counter/PC⟨15:0⟩ := R[7]⟨15:0⟩
: points to the current instruction being interpreted. It will be seen that the fact that PC is one of the general registers is crucial to the design.

Any general register, R[0:7], can be used as a stack pointer. The special Stack Pointer (SP) has additional properties that force it to be used for changing processor state interrupts, traps, and subroutine calls (It also can be used to control dynamic temporary storage subroutines.)

In addition to the above registers there are 8 bits used (from a possible 16) for processor status, called PS⟨15.0⟩ register. Four bits are the Condition Codes (CC) associated with arithmetic results; the T-bit controls tracing; and three bits control the priority of running programs Priority ⟨2:0⟩. Individual bits are mapped in PS as shown in Appendix 1.

Data types and primitive operations

There are two data lengths in the basic machine: bytes and words, which are 8 and 16 bits, respectively. The non-trivial data types are word length integers (w.i.); byte length integers (by .i); word length boolean vectors (w.bv), i.e., 16 independent bits (booleans) in a 1 dimensional array; and byte length boolean vectors (by.bv). The operations on byte and word boolean vectors are identical. Since a common use of a byte is to hold several flag bits (booleans), the operations can be combined to form the complete set of 16 operations. The logical operations are: "clear," "complement," "inclusive or," and "implication" ($x \supset y$ or $\neg x \vee y$).

There is a complete set of arithmetic operations for the word integers in the basic instruction set. The arithmetic operations are: add, subtract, multiply (optional), divide (optional), compare, add one, subtract one, clear, negate, and multiply and divide by powers of two (shift). Since the address integer size is 16 bits, these data types are most important. Byte length integers are operated on as words by moving them to the general registers where they take on the value of word integers. Word length integer operations are carried out and the results are returned to memory (truncated).

The floating point instructions defined by software (not part of the basic instruction set) require the definition of two additional data types (of length two and three), i.e., double word (d.w.) and triple (t.w.) words. Two additional data types, double integer (d.i.) and triple floating point (t.f. or f) are provided for arithmetic. These data types imply certain additional operations and the conversion to the more primitive data types.

Address (operand) calculation

The general methods provided for accessing operands are the most interesting (perhaps unique) part of the machine's structure. By defining several access methods to a set of general registers, to memory, or to a stack (controlled by a general register), the computer is able to be a 0, 1 and 2 address machine. The encoding of the instruction Source (S) fields and Destination (D) fields are given in Fig. 10 together with a list of the various access modes that are possible. (Appendix 1 gives a formal description of the effective address calculation process.)

It should be noted from Figure 10 that all the common access modes are included (direct, indirect, immediate, relative, indexed, and indexed indirect) plus several relatively uncommon ones. Relative (to PC) access is used to simplify program loading, while immediate mode speeds up execution. The relatively uncommon access modes, auto-increment and auto-decrement, are used for two purposes: access to a stack under control of the registers* and access to bytes or words organized as strings or vectors. The indirect access mode allows a stack to hold addresses of data (instead of data). This mode is desirable when manipulating longer and variable-length data types (e.g., strings, double fixed and triple floating point). The register auto increment mode may be used to access a byte string; thus, for example, after each access, the register can be made to point to the next data item. This is used for moving data blocks, searching for particular elements of a vector, and byte-string operations (e.g., movement, comparisons, editing).

* A definition of the ISP notation used here may be found in Appendix 1.

*Note, by convention a stack builds toward register 0, and when the stack crosses 400_8, a stack overflow occurs.

The assignment of bus mastership is done concurrent with normal communication (dialogues).

Unibus dialogues

Three types of dialogues use the Unibus. All the dialogues have a common protocol which first consists of obtaining the bus mastership (which is done concurrent with a previous transaction) followed by a data exchange with the requested device. The dialogues are: Interrupt; Data In and Date In Pause; and Data Out and Data Out Byte.

Interrupt

Interrupt can be initiated by a master immediately after receiving bus mastership. An address is transmitted from the master to the slave on Interrupt. Normally, subordinate control devices use this method to transmit an interrupt signal to the processor.

Data in and data in pause

These two bus operations transmit slave's data (whose address is specified by the master) to the master. For the Data In Pause operation data is read into the master and the master responds with data which is to be rewritten in the slave.

Data out and data out byte

These two operations transfer data from the master to the slave at the address specified by the master. For Data Out a word at the address specified by the address lines is transferred from master to slave. Data Out Byte allows a single data byte to be transmitted.

Processor logical design

The Pc is designed using TTL logical design components and occupies approximately eight 8″ × 12″ printed circuit boards. The organization of the logic is shown in Figure 17. The Pc is physically connected to two other components, the console and the Unibus. The control for the Unibus is housed in the Pc and occupies one of the printed circuit boards. The most regular part of the Pc, the arithmetic and state section, is shown at the top of the figure. The 16-word scratch-pad memory and combinatorial logic data operators, D(shift) and D(adder, logical ops), form the most regular part of the processor's structure. The 16-word memory holds most of the 8-word processor state found in the ISP, and the 8 bits that form the Status word are stored in an 8-bit register. The input to the adder-shift network has two latches which are either memories or gates. The output of the adder-shift network can be read to either the data or address parts of the Unibus, or back to the scratch-pad array.

The instruction decoding and arithmetic control are less regular than the above data and state and these are shown in the lower part of the figure. There are two major sections: the instruction fetching and decoding control and the instruction set interpreter (which in effect defines the ISP). The later control section operates on, hence controls, the arithmetic and state parts of the Pc. A final control is concerned with the interface to the Unibus (distinct from the Unibus control that is housed in the Pc).

CONCLUSIONS

In this paper we have endeavored to give a complete description of the PDP-11 Model 20 computer at four descriptive levels. These present an unambiguous specification at two levels (the PMS structure and the ISP), and, in addition, specify the constraints for the design at the top level, and give the reader some idea of the implementation at the bottom level logical design. We have also presented guidelines for forming additional models that would belong to the same family.

ACKNOWLEDGMENTS

The authors are grateful to Mr. Nigberg of the technical publication department at DEC and to the reviewers for their helpful criticism. We are especially grateful to Mrs. Dorothy Josephson at Carnegie-Mellon University for typing the notation-laden manuscript.

REFERENCES

1 R H ALLMARK J R LUCKING
Design of an arithmetic unit incorporationg a nesting store
Proc IFIP Congress pp 694–698 1962

2 G M AMDAHL G A BLAAUW F P BROOKS JR
Architecture of the IBM System/360
IBM Journal Research and Development Vol 8 No 2 pp 87–101 April 1964

3 C G BELL A NEWELL
Computer structures
McGraw-Hill Book Company Inc New York In press 1970

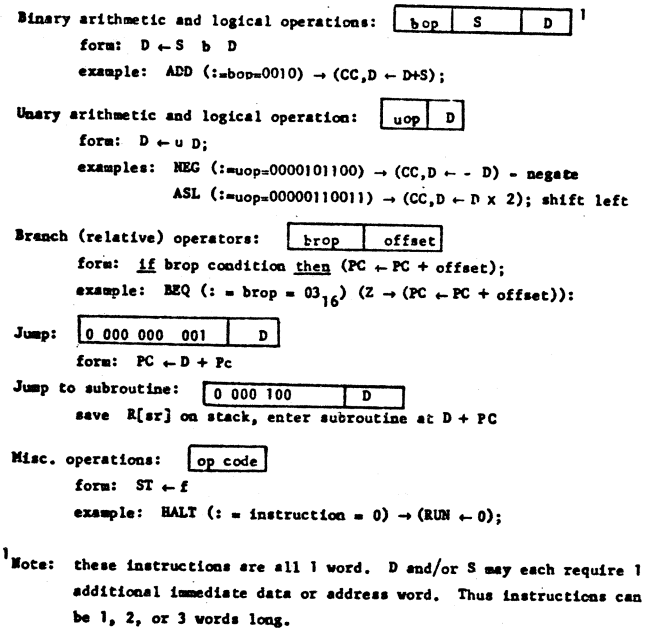


Figure 12—PDP-11 instruction formats (simplified)

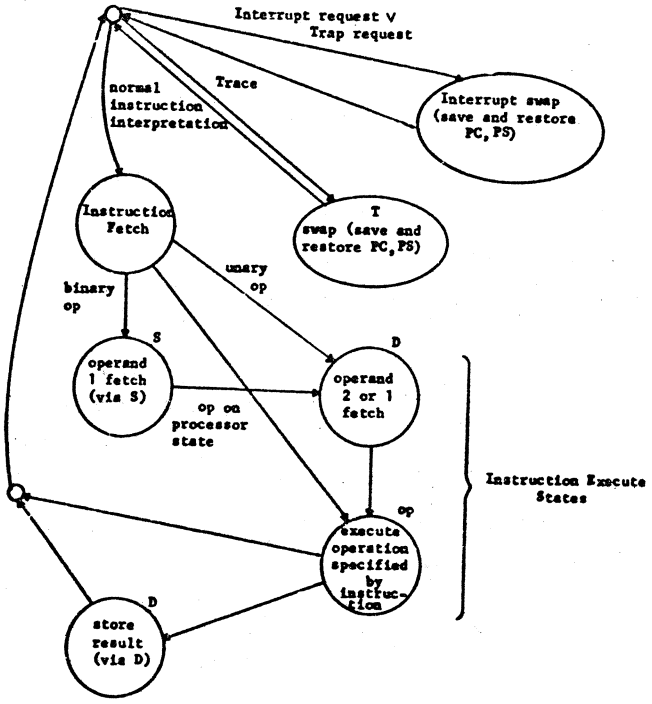


Figure 13—PDP-11 instruction interpretation process state diagram

trap occurs after each instruction, and (3) normal instruction interpretation. The five (lower) states in the diagram are concerned with instruction fetching, operand fetching, executing the operation specified by the instruction and storing the result. The non-trivial details for fetching and storing the operands are not shown in the diagram but can be constructed from the effective address calculation process (Appendix 1). The state diagram, though simplified, is similar to 2- and 3-address computers, but is distinctly different than a 1 address (1 accumulator) computer.

The ISP description (Appendix 1) gives the operation of each of the instructions, and the more conventional diagram (Fig. 12) shows the decoding of instruction classes. The ISP description is somewhat incomplete; for example, the add instruction is defined as: ADD (:= bop = 0010) → (CC,D ← D + S); *addition* does not exactly describe the changes to the Condition Codes/CC (which means whenever a binary opcode [bop] of 0010_2 occurs the ADD instruction is executed with the above effect). In general, the CC are based on the result, that is, Z is set if the result is zero, N if negative, C if a carry occurs, and V if an overflow was detected as a result of the operation. Conditional branch instructions may thus follow the arithmetic instruction to test the results of the CC bits.

Examples of addressing schemes

Use as a stack (zero address) machine

Figure 14 lists typical zero-address machine instructions together with the PDP-11 instructions which perform the same function. It should be noted that translation (compilation) from normal infix expressions to reverse Polish is a comparatively trivial task. Thus, one of the primary reasons for using stacks is for the evaluation of expressions in reverse Polish form.

Consider an assignment statement of the form

$$D \leftarrow A + B/C$$

which has the reverse Polish form

$$DABC/+ \leftarrow$$

and would normally be encoded on a stack machine as follows

load stack address of D
load stack A
load stack B
load stack C
/
+
store

```
PS⟨15:0⟩                                          (processor state register)
  Priority/P⟨2:0⟩ := PS⟨7:5⟩                      (under program control; priority
                                                   level of the process currently being
                                                   interpreted a higher level process
                                                   may interrupt or trap this process)
  CC/Condition␣Codes⟨3:0⟩ := PS⟨3:0⟩               (under program control; when set,
                                                   each instruction executed will trap;
                                                   used for interpretive and break-
                                                   point debugging)
    Carry/C := CC⟨0⟩                              (a result condition code indicating
                                                   an arithmetic carry from bit 15 of
                                                   the last operation)
    Negative/N := CC⟨3⟩                           (a result condition code indicating
                                                   last result was negative)
    Zero/Z := CC⟨2⟩                               (a result condition code indicating
                                                   last result was zero)
    Overflow/V := CC⟨1⟩                           (a result condition code indicating
                                                   an arithmetic overflow of the last
                                                   operation)
    Trace/T := ST⟨4⟩                              (denotes whether instruction trace
                                                   trap is to occur after each instruc-
                                                   tion is executed)
    Undefined⟨7:0⟩ := PS⟨15:8⟩                    (unused)
  Run                                             (denotes normal execution)
  Wait                                            (denotes waiting for an interrupt)

Instruction Format
(Bit assignments used in the various instruction formats)
  i/instruction⟨15:0⟩
    bop⟨3:0⟩ := i⟨15:12⟩                          (binary operation code)
    uop⟨15:6⟩ := i⟨15:6⟩                          (unary operation code)
    brop⟨15:8⟩ := i⟨15:8⟩                         (branch operation code)
    sop⟨15:6⟩ := i⟨15:6⟩                          (shift operation code)
    s/source⟨5:0⟩ := i⟨11:6⟩                      (source control byte)
      sm⟨0:1⟩ := s⟨5:4⟩                           (source mode control)
      sd      := s⟨3⟩                             (source defer bit)
      sr      := s⟨2:0⟩                           (source register)
    d/destination⟨5:0⟩ := i⟨5:0⟩                  (destination control byte)
      dm⟨0:1⟩ := d⟨5:4⟩
      dd      := d⟨3⟩
      dr⟨2:0⟩ := d⟨2:0⟩
    offset⟨7:0 := i⟨7:0⟩                          (signed 7 bit integer)
  address␣increment/ai                            (implicit bit derived from i to denote
                                                   byte or word length operations)

Data Types
  by/byte⟨7:0⟩
  w/word⟨15:0⟩                                    (signed integers)
  by.i/byte.integer⟨7:0⟩
  w.i/word.integer⟨15:0⟩
  by.bv/byte.boolean␣vector⟨7:0⟩                  (boolean vectors (bits))
  w.bv/word.boolean␣vector⟨15:0⟩
```

(32 bit) registers, D. In order to access the various parts of F or D registers, registers F0 and F1 are mapped onto registers R0 to R2 and R3 to R5.

Since the instruction set operation code is almost completely encoded already for byte and word length data, a new encoding scheme is necessary to specify the proposed additional instructions. This scheme adds two instructions: enter floating point mode and execute one floating point instruction. The instructions for floating point and double word data would be:

binary ops	op	floating point/f	and double word/d
bop′ S D	←	FMOVE	DMOVE
	+	FADD	DADD
	−	FSUB	DSUB
	×	FMUL	DMUL
	/	FDIV	DDIV
	compare	FCMP	DCMP
unary ops			
uop′ D	−	FNEG	DNEG

LOGICAL DESIGN OF S(UNIBUS) AND PC

The logical design level is concerned with the physical implementation and the constituent combinatorial and sequential logic elements which form the various computer components (e.g., processors, memories, controls). Physically, these components are separate and connected to the Unibus following the lines of the PMS structure.

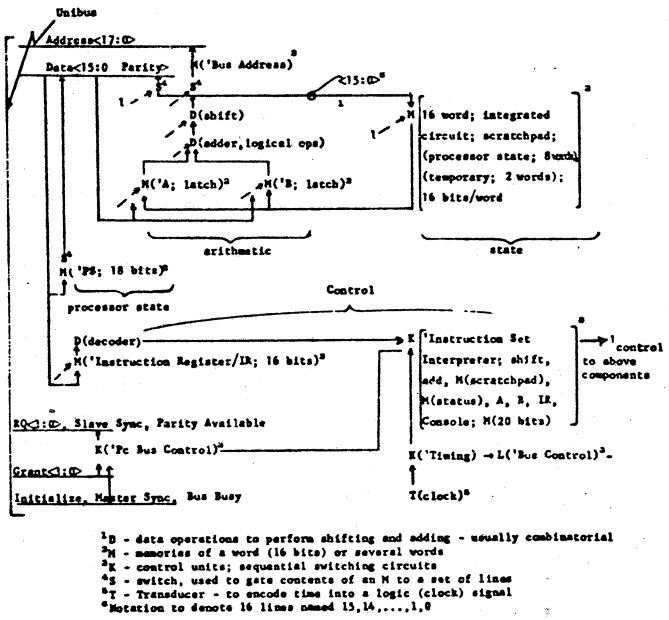
Unibus organization

Figure 16 gives a PMS diagram of the Pc and the entering signals from the Unibus. The control unit for the Unibus, housed in Pc for the Model 20, is not shown in the figure.

The PDP-11 Unibus has 56 bi-directional signals conventionally used for program-controlled data transfers (processor to control), direct-memory data transfers (processor or control to memory) and control-to-processor interrupt. The Unibus is interlocked; thus transactions operate independent of the bus length and response time of the master and slave. Since the bus is bi-directional and is used by all devices, any device can communicate with any other device. The controlling device is the master, and the device to which the master is communicating is the slave. For example, a data transfer from processor (master) to memory (always a slave) uses the Data Out dialogue facility for writing and a transfer from memory to processor uses the Data In dialogue facility for reading.

Bus control

Most of the time the processor is bus master fetching instructions and operands from memory and storing results in memory. Bus mastership is determined by the current processor priority and the priority line upon which a bus request is made and the physical placement of a requesting device on the linked bus.



[1]D - data operations to perform shifting and adding - usually combinatorial
[2]M - memories of a word (16 bits) or several words
[3]K - control units; sequential switching circuits
[4]S - switch, used to gate contents of an M to a set of lines
[5]T - Transducer - to encode time into a logic (clock) signal
[6]Notation to denote 16 lines named 15,14,...,1,0

Figure 16—PDP-11 Pc structure

Binary Arithmetic: D ← D b S;

ADD(:= bop = 0110) → (CC,D ← D+s);	(*add*)
SUB(:= bop = 1110) → (CC,D ← D − S);	(*subtract*)
CMP(:= bop = 0010) → (CC ← D − S);	(*word compare*)
CMPB(:= bop = 1010) → (CC ← Db − Sb);	(*byte compare*)
MUL(:= bop = 0111) → (CC,D ← D × S);	(**multiply if D is a register then a double length operator*)
DIV(:= bop = 1111) → (CC,D ← D/S);	(**divide, if D is a register, then a remainder is saved*)

Unary Arithmetic D ← u S;

CLR(:= uop = 050_8) → (CC,D ← 0);	(*clear word*)
CLRB(:= uop = 1050_8) → (CC,Db ← 0);	(*clear byte*)
COM(:= uop = 051_8) → (CC,D ← ¬D);	(*complement word*)
COMB(:= uop = 1051_8) → (CC,Db ← ¬Db);	(*complement byte*)
INC(:= uop = 052_8) → (CC,D ← D + 1);	(*increment word*)
INCB(:= uop = 1052_8) → (CC,Db ← Db + 1);	(*increment byte*)
DEC(:= uop = 053_8) → (CC,D ← D − 1);	(*decrement word*)
DECB(:= uop = 1053_8) → (CC,Db ← Db − 1);	(*decrement byte*)
NEG(:= uop = 054_8) → (CC,D ← −D);	(*negate*)
NEGB(:= uop = 1054_8) → (CC,Db ← −Db)	(*negate byte*)
ADC(:= uop = 055_8) → (CC,D ← D + C);	(*add the carry*)
ADCB(:= uop = 1055_8) → (CC,Db ← Db + C);	(*add to byte the carry*)
SBC(:= uop = 056_8) → (CC,D ← D − C);	(*subtract the carry*)
SBCB(:= uop = 1056_8) → (CC,Db ← Db − C);	(*subtract from byte the carry*)
TST(:= uop = 057_8) → (CC ← D);	(*test*)
TST(:= uop = 1057_8) → (CC ← Db);	(*test byte*)

Shift operations: D ← D × 2^n;

ROR(:= sop = 060_8) → (C□D ← C□D/2{rotate};	(*rotate right*)
RORB(:= sop = 1060_8) → (C□Db ← C□Db/2{rotate});	(*byte rotate right*)
ROL(:= sop = 061_8) → (C□D ← C□D × 2{rotate});	(*rotate left*)
ROLB(:= sop = 1061_8) → (C□Db ← C□Db × 2{rotate});	(*byte rotate left*)
ASR(:= sop = 062_8) → (CC,D ← D × 2);	(*arithmetic shift right*)
ASRB(:= sop = 1062_8) → (CC,Db ← Db/2);	(*byte arithmetic shift right*)
ASL(:= sop = 063_8) → (CC,D ← D × 2);	(*arithmetic shift left*)
ASLB(:= sop = 1063_8) → (CC,Db ← Db × 2);	(*byte arithmetic shift left*)
ROT(:= sop = 064_8) → (C□D ← D × 2^s);	(*rotate*)
ROTB(:= sop = 1064_8) → (C□Db ← D × 2^s);	(*byte rotate*)
LSH(:= sop = 065_8) → (CC,D ← D × 2^s{logical});	(**logical shift*)
LSHB(:= sop = 1065_8) → (CC,Db ← Db × 2^s{logical});	(**byte logical shift*)
ASH(:= sop = 066_8) → (CC,D ← D × 2^s);	(**arithmetic shift*)
ASHB(:= sop = 1066_8) → (CC,Db ← Db × 2^s);	(**byte arithmetic shift*)
NOR(:= sop = 067_8) → (CC,D ← normalize(D)); (R[r′] ← normalize␣exponent(D));	(**normalize*)
NORD(:= sop = 1067_8) → (Db ← normalize(Dd); R[r′] ← normalize␣exponent(D));	(**normalize double*)
SWAB(:= sop = 3) → (CC,D ← D⟨7:0, 15:8⟩)	(*swap bytes*)

Logical Operations

BIC(:= bop = 0100) → (CC,D ← D ← D ∧ ¬S);	(*bit clear*)
BICB(:= bop = 1100) → (CC,Db ← Db ∨ ¬Sb);	(*byte bit clear*)
BIS(:= bop = 0101) → (CC,D ← D ∨ S);	(*bit set*)
BISB(:= bop = 1101) → (CC,Db ← Db ∨ Sb);	(*byte bit set*)
BIT(:= bop = 0011) → (CC ← D ∧ S);	(*bit test under mask*)
BITB(:= bop = 1011) → (CC ← Db ∧ Sb);	(*byte bit test under mask*)

4 A W BURKS H H GOLDSTINE J VON NEUMANN
Preliminary discussion of the logical design of an electronic computing instrument, Part II
Datamation Vol 8 No 10 pp 36–41 October 1962
5 W S ELLIOTT C E OWEN C H DEVONALD B G MAUDSLEY
The design philosophy of Pegasus, a quantity-production computer
Proceedings IEEE Pt. B 103 Supp 2 pp 188–196 1956
6 F M HANEY
Using a computer to design computer instruction sets
Thesis for Doctor of Philosophy degree College of Engineering and Science Department of Computer Science Carnegie-Mellon University Pittsburgh Pennsylvania May 1968
7 W LONERGAN P KING
Design of the B5000 system
Datamation Vol 7 No 5 pp 28–32 May 1961
8 W D MAURER
A theory of computer instructions
Journal of the ACM Vol 13 No 2 pp 226–235 April 1966
9 S ROTHMAN
R/W 40 data processing system
International Conference on Information Processing and Auto-math 59 Ramo-Wooldridge (A division of Thompson Ramo Wooldridge Inc) Los Angeles California June 1959
10 M V WILKES
The best way to design an automatic calculating machine
Report of Manchester University Computer Inaugural Conference July 1951 (Manchester 1953)

APPENDIX 1

DEC PDP-11 instruction set processor Description (in ISPL*)

The following description is not a detailed description of the instructions. The description omits the trap behavior of unimplemented instructions, references to non-existent primary memory and io devices, SP (stack) overflow, and power failure.

Primary Memory State

M/Mb/Memory[0:$2^{16}-1$]⟨7:0⟩	*(byte memory)*
Mw[0:$2^{15}-1$]⟨15:0⟩ : = M[0:$2^{16}-1$]⟨7:0⟩	*(word memory mapping)*

Processor State (9 words)

R/Registers[0:7]⟨15:0⟩	*(word general registers)*
SP⟨15:0⟩ : = R[6]⟨15:0⟩	*(stack pointer)*
PC⟨15:0⟩ : = R[7]⟨15:0⟩	*(program counter)*

---

*ISP NOTATION

Although the ISP language has not been described in publications, its syntax is similar to other languages. The language is inherently interpreted in parallel, thus to get sequential evaluation the word "next" must be used. Italics are used for comments. The following notes are in order:

$a := f(\ldots)$	equivalence or substitution process used for name and process substitution. For every occurrence of $a, f(\ldots)$ replaces it.
$a \leftarrow f(\ldots)$	Replacement operator; the contents in register a are replaced by the value of the function.
register declaration, e.g., $Q[0:1]\,[0:4095]\,\langle 15:0\rangle$	an array of words of two dimensions 2 and 4096; each word has 16 bits denoted 15, 14, 13, . . ., 1, 0
$\langle a:b\rangle_n$	Denotes a range of characters $a, a+1, \ldots, b$ to base n. If n is not given, the base is 2.
$[c:d]$	Array designation $c, c+1, \ldots, d$
$a \rightarrow b;$	equivalent to ALGOL if a then b
"next"	sequential interpretation
instruction declaration, e.g., ADD (: = bop = 0010) → (*CC*, $D \leftarrow D + S$)	defines the "ADD" instruction, assigns it a value, and gives its operation. ADD is executed when bop = 0010_2. Equivalent to: ADD → (*CC*, $D \leftarrow D + S$) where ADD: = (bop = 0010) bop has been previously declared
□	concatenation, consider the combined registers as one

operators: = (+/add | −/subtract/negate | ×/multiply | //divide | ∧/and | ∨/or | √/not | ⊕/exclusive or | =/equal/>/greater than | ≥ | < | ≤ | ≠ | modulo | etc.)

```
d/double␣word⟨31:0⟩                                   (*double word)
t/triple␣word⟨47:0⟩                                   (*triple word)
f/t.f/triple.floating␣point⟨47:0⟩                     (*triple floating point)

Source/S and Destination/D Calculation
  S/Source⟨15:0⟩ : = (¬ sd → (                        (direct access)
    (sm = 00) → R[sr];                                (register)
    (sm = 01) ∧ (sr ≠ 7) → (M[R[sr]]; next R[sr] ← R[sr] + ai);   (auto increment)
    (sm = 01) ∧ (sr = 7) → (M[PC]; PC ← PC + 2);      (immediate)
    (sm = 10) → (R[sr] ← R[sr] − ai; next M[R[sr]]);  (auto decrement)
    (sm = 11) ∧ (sr ≠ 7) → (M[M[PC] + R[sr]]; PC ← PC + 2);   (indexed)
    (sm = 11) ∧ (sr = 7) → (M[M[PC] + PC]; PC ← PC + 2));     (relative)
                    sd → (                            (indirect access)
    (sm = 00) → M[R[sr]];                             (indirect via register)
    (sm = 01) ∧ (sr ≠ 7) → (M[M[R[sr]]]; next R[sr] ← R[sr] + ai);   (indirect via stack, auto decrement)
    (sm = 01) ∧ (sr = 7) → (M[M[PC]]; PC ← PC + 2);   (direct absolute)
    (sm = 10) → (R[sr] ← R[sr] − ai; next M[R[sr]]);  (indirect via stack, auto increments)
    (sm = 11) ∧ (sr ≠ 7) → (M[M[PC] + R[sr]]; PC ← PC + 2);   (indirect, indexed)
    (sm = 11) ∧ (sr = 7) → (M[M[M[PC] + PC]]; PC ← PC + 2))   (indirect relative)
```

(The above process defines how operands are determined (accessed) from either memory or the registers. The various length operands, Db(byte), Dw(word), Dd(double) and Df(floating) are not completely defined. The Source/S and Destination/D processes are identical. In the case of jump instruction an address, D′, is used—instead of the word in location M[CI].)

```
Instruction Interpretation Process
  ¬ Interrupt␣rqs ∧ Run ∧ Wait → (i ← M[PC]; PC ← PC + 2;        (fetch)
                                  next instruction␣execution; next   (execute)
    T → (SP ← SP + 2; next                                      (trace bit store state)
      M[SP] ← PS;
      SP ← SP + 2; next
      M[SP] ← PC;
      PC ← M[14₈]
      ST ← M[16₈]))

  Interrupt␣rq[j] ∧ (CC[j] > CC) ∧ Run → (T ← 0;                (interrupt)
    SP ← SP + 2; next
    M[SP] ← PS;                                                 (store state and PC enter new process). The locations M[f(j)] are:
                                                                reserved instruction = M[10]
                                                                illegal instruction = M[4]
                                                                stack overflow = M[4]
                                                                bus errors = M[4])
    SP ← SP + 2;
    M[SP] ← PC
    PC ← M[f(j)]
    PS ← M[f(j) + 2])
Instruction Set and the Execution Process
```

(The following instruction set will be defined briefly and is incomplete. It is intended to give the reader a simple understanding of the machine operation.)

```
Instruction␣execution : = (
  MOV(: = bop = 0001) → (CC,D ← S);                             (move word)
  MOVB(: = bop = 1001) → (CC,Db ← Sb);                          (move byte)
```

* not hardwired or optional

```
Branches and Subroutines Calling: PC ← f;
  JMP(:= sop = 0001₈) → (PC ← D');                          (jump unconditional)
  BR(:= brop = 01₁₆) → (PC ← PC + offset);                  (branch unconditional)
  BEQ(:= brop = 03₁₆) → (Z → (PC ← PC + offset));           (equal to zero)
  BNE(:= brop = 02₁₆) → (¬Z → (PC ← PC + offset));          (not equal to zero)
  BLT(:= brop = 05₁₆) → (N ⊕ V → (PC ← PC + offset));       (less than (zero))
  BGE(:= brop = 04₁₆) → (N ≡ V → (PC ← PC + offset));       (greater than or equal (zero))
  BLE(:= brop = 07₁₆) → (Z ∨ (N ⊕ V) → (PC ← PC + offset)); (less than or equal (zero))
  BGT(:= brop = 06₁₆) → (¬(Z ∨ (N ⊕ V)) → (PC ← PC + offset)); (less greater than (zero))
  BCS/BHIS(:= brop = 87₁₆) → (C → (PC ← PC + offset));      (carry set; higher or same (unsigned))

  BCC/BLO(:= brop = 86₁₆) → (¬C → (PC ← PC + offset));      (carry clear; lower (unsigned))
  BLOS(:= brop = 83₁₆) → (C ∧ Z → (PC ← PC + offset));      (lower or same (unsigned))
  BHI(:= brop = 82₁₆) → ((¬C ∨ Z) → (PC ← PC + offset));    (higher than (unsigned))
  BVS(:= brop = 85₁₆) → (V → (PC ← PC + offset));           (overflow)
  BVC(:= brop = 84₁₆) → (¬V → (PC ← PC + offset));          (no overflow)
  BMT(:= brop = 81₁₆) → (N → (PC ← PC + offset));           (minus)
  BPL(:= brop = 80₁₆) → (¬N → (PC ← PC + offset));          (plus)
  JSR(:= sop = 004Ø₈) → (                                   (jump to subroutine by putting
    SP ← SP − 2; next                                        R[sr], PC on stack and loading
    M[SP] ← R[sr];                                           R[sr] with PC, and going to sub-
    R[sr] ← PC;                                              routine at D)
      PC ← D);
  RTS(:= i = 00020Ø₈) → (                                   (return from subroutine)
    PC ← R[dr];
    R[dr] ← M[SP];
    SP ← SP + 2);

Miscellaneous processor state modification:
  RTI(:= i = 2₈) → (PC ← M[SP];                             (return from interrupt)
                    SP ← SP + 2; next
                    PS ← M[SP];
                    SP ← SP + 2);
  HALT(:= i = 0) → (Run ← 0);
  WAIT(:= i = 1) → (Wait ← 1);
  TRAP(:= i = 3) → (SP ← SP + 2; next                       (trap to M[34₈] store status and
                    M[SP] ← PS;                              PC)
                    SP ← SP + 2; next
                    M[SP] ← PC;
                    PC ← M[34₈];                            (enter new process)
                    PS ← M[12]);
  EMT(:= brop − 82₁₆) → (                                   (emulator trap)
                      SP ← SP + 2; next
                      M[SP] ← PS;
                      SP ← SP + 2; next
                      M[SP] ← PC;
                      PC ← M[30₈];
                      PS ← M[32₈]);
IOT(:= i = 4) → (see TRAP)                                  (I/O trap to M[20₈])
  RESET(:= i = 5) → (not described)                         (reset to external devices)
  OPERATE(:= i⟨5:15⟩ = 5) → (                               (condition code operate)
    i⟨4⟩ → (CC ← CC ∨ i⟨3:0⟩);                              (set codes)
    ¬i⟨4⟩ → (CC ← CC ∧ ¬ i⟨3:0⟩)));                         (clear codes)
                                  end Instruction␣execution
```

digital

PDP-11/10

WHY DIGITAL?

The DIGITAL story is singular in the computer industry. Founding the minicomputer market in 1963, DIGITAL now has over 20,000 computer installations throughout the world, over 100 sales offices, and seven factories in five countries.

The product breadth of the company is also unequaled. Small, medium, or large-scale computer systems vary in price from under $4000 to several million dollars. A full line of peripheral equipment is also available. Most of the line is DIGITAL-designed and built and is therefore processor compatible and manufactured and tested to rigid specifications.

DIGITAL is also the world's leading manufacturer of circuit modules, providing industrial control modules, analog and digital conversion equipment, and high-speed computer modules.

In addition to standard products, DIGITAL provides design services where products can be designed and fabricated to customer specifications....And the company is still growing at a rapid rate, averaging over 12 new products per month.

With DIGITAL, a customer need only look to a single supplier...a supplier with the knowhow, production facilities and service organization to meet almost every computing requirement.

PRODUCT SERVICING

DIGITAL's success is due in large measure to the company's field service organization with its more than 1000 engineers in 100 world-wide service centers.

It is this service organization that tests each system prior to shipment, installs the system, and through a large variety of contract types, assures its continued operation.

Services range from standard 8-hour shifts to full 24-hour coverage in contracts that include preventive maintenance as well as repairs. In resident service, an engineer is located full-time at the customer's site. And for less demanding requirements, service is available on a charge-per-call basis.

In addition to equipment servicing, DIGITAL provides software specialists to handle customer on-site training, application program assistance, and service calls, so that software problems receive fast, efficient service.

TRAINING

Formalized software and hardware training is available to every DIGITAL customer. Full time training staffs conduct regularly scheduled courses at training centers in Maynard, Massachusetts; Sunnyvale, California; Paris, France; Munich, Germany; and Reading, England.

By special arrangement, courses can be designed to meet specific requirements.

OTHER SERVICES

For special design requirements, DIGITAL provides the following. A Computer Special Systems Group designs non-standard system equipment to customer specifications. A System Engineering Group specializes in the development of multi-processor networks. And a Control Products Group develops unique control interfaces and devices employing DIGITAL-designed and fabricated logic modules.

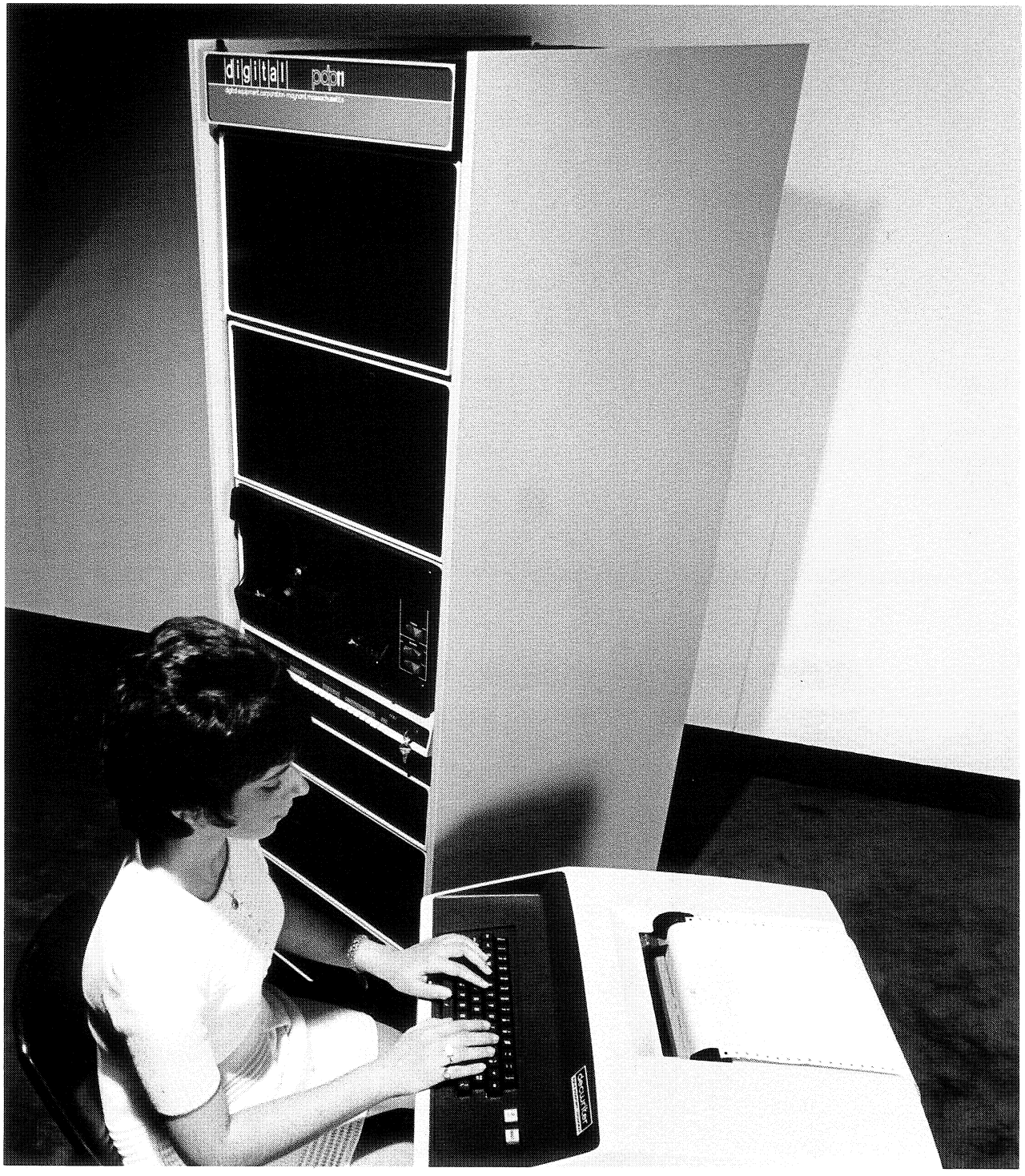
DECUS, the Digital Equipment Computer Users Society —the most active users group in the world—is designed to stimulate the exchange of information and help solve common problems. Representing some 10,000 users from more than 40 countries, the society sponsors an active newsletter and operates periodic local and national symposia. The society also maintains an invaluable library of user-contributed programs.

CUSTOMER APPLICATION GROUPS

The following DIGITAL groups service specialized areas of customer interest. An Industrial Products Group handles industrial control and data acquisition applications. The Laboratory Data Products Group is responsible for laboratory systems in the physical and life sciences. A Medical Systems Group specializes in products for hospitals and diagnostic clinics. A Graphics Arts Group supplies computerized typesetting systems and other graphic products to the newspaper and publishing community. An Education Group specializes in systems for secondary schools, junior colleges, and universities. A Data Systems group handles commercial data processing systems. A Communications Group supplies products that meet a wide variety of data communications problems.

Separate groups also handle the needs of Original Equipment Manufacturers (OEM's).

PDP-11/10... A SMALL PACKAGE WITH LARGE CAPABILITIES



The economy computer, like the economy car, is usually cheap because it's stripped down; by the time the necessary options are added, price can double.

Although PDP-11/10 is economical in price, it is not one of these stripped down models. It is a full fledged PDP-11 family member—designed around the flexible UNIBUS™ structure—with the same instruction set and byte word addressing structure as the popular PDP-11/20. Included are such standard features as a teletypewriter interface, 8 general registers, hardware stacks, a hardware interrupt system with direct vectors for fast interrupt handling.

The system also includes a real-time clock and power fail protection and automatic restart—features which are extremely valuable in industrial and laboratory applications.

The economy of the PDP-11/10 is due to a totally new packaging design. The basic processor with 8 K words of memory is only 5-1/4 inches high and will fit unobtrusively almost anywhere: underneath a terminal, inside control machinery, in a desk drawer, or in a suitcase.

Unlike other systems, however, the system's small size and low cost do not preclude its expansion. The system can expand to include up to 28 K words of memory, a large number of peripherals, including disk, and is completely compatible with larger PDP-11 family members. For example, programs written on the PDP-11/20 can run on the PDP-11/10 with no modifications. And if and when upgrading is necessary, a larger PDP-11 can replace the PDP-11/10 with plug-in ease.

The PDP-11/10 is ideal for applications with limited storage or I/O requirements or for such dedicated applications as communications front end processing or a timesharing terminal controller. These applications may be in conjunction with a larger PDP-11 or with another larger processor such as the DECsystem-10.

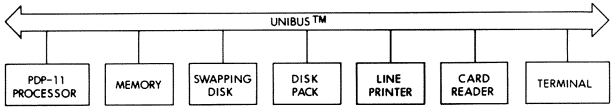
In short, the PDP-11/10 is a powerful, economical and compatible processor. For the cost conscious user, it represents the most cost effective system available today...one which can be easily upgraded if the need ever arises.

UNIBUS ARCHITECTURE

A key to the PDP-11 family's many strengths is the fact that all system elements—processor, memory, peripherals—plug into a single asynchronous high speed bus. Known as a UNIBUS, this bidirectional bus provides easy interfacing and simplifies the construction of multiprocessor or shared peripheral configurations.

UNIBUS architecture keeps PDP-11 systems from becoming obsolete. Due to its asynchronous nature, the UNIBUS is compatible with devices that operate over a wide range of speeds. Therefore, faster devices or memory can always replace older versions without obsoleting the system.

With the UNIBUS, fast devices have easy direct memory access—no multiplexers or synchronizing DMA hardware is required. These devices can send, receive, or exchange data without processor intervention and without intermediate buffering in memory. Transfers on the bus take place at rates up to 2.5 million words per second.



CORE MEMORY

Expanding the PDP-11/10 is as easy as plugging in modules. Read/write core memory is available in 8 K word modules of 900 nanosecond speed which can be interleaved to achieve faster memory cycle times. Maximum system memory of up to 28 K words can be comprised of DIGITAL-manufactured memory of different speeds and characteristics so that the new memory can always be added to, or replace, the old.

Bootstrap loaders—32 words of read only memory—are available for automatic loading of disk, DECtape, and paper tape. These options eliminate manual key-in of startup data and are extremely useful when the system is being operated by non-computer oriented personnel.

A special uncoded loader lets the user prepare a bootstrap for a special device; he merely clips unneeded diodes from the circuitry.

INSTRUCTION SET

The PDP-11's comprehensive instruction set provides the programming flexibility of a large computer in a 16-bit mainframe. The set provides unusual but often required instructions, so that a single instruction often suffices where several may be required in a traditional machine. For example, a bit test instruction (BIT) can test any bit or combination of bits to determine their state. In a conventional machine, this task would require several masking operations.

Bit, byte, and word addressing in both single and double operand formats make possible memory saving and simplify the implementation of control and communications applications.

PDP-11 double operand instructions allow a programmer to perform several operations with a single instruction. For example, ADD A, B adds the contents of location A to location B and stores the result in location B. With the traditional instruction set, three instructions would be required (see example).

PDP-11

ADD A, B Add contents of location A to location B and store results in location B.

Conventional

LDA A Load contents of location A into accumulator.
ADD B Add contents of location B into accumulator.
STA B Store results in location B.

BUS

PDP-11

Single asynchronous bus, the UNIBUS, means greater system efficiency, easier interfacing and expansion.

CONVENTIONAL

Two or more synchronous buses, separate for I/O and memory.

Due to the system's UNIBUS, the PDP-11 does not require special I/O instructions; the same instruction that performs a register-to-register transfer performs a memory-to-device register transfer or a memory-to-memory transfer. In the double operand instruction ADD A, B, therefore, A and B can be registers, a register and a memory location or two memory locations.

The instruction set contains a full set of conditional branches, eliminating excessive use of "jump" instructions. A branch is also included for overflow of a signed integer. Many computers over-economize and eliminate this branch therefore requiring a 5 to 8 instruction subroutine to perform this important function.

All instructions can directly address the full 32 K word memory and I/O space.

8 GENERAL REGISTERS

With 8 general registers, the PDP-11 gives the programmer the flexibility and speed of a large-scale computer.

Most small computers have 2 to 4 registers which are generally used as accumulators and sometimes used as index registers. Since PDP-11 registers are not dedicated to specific functions, the programmer can assign them dynamically, depending on whether he needs to manipulate a pointer, achieve speed, or use the registers for temporary storage.

Where speed is the criterion, register-to-register operations can be performed. The PDP-11 also allows direct memory-to-register operations. For example, the programmer can add from memory to a register without having to first load one register then add into another register, as with conventional systems.

Registers can also be reserved for time critical partial results or pointer functions. The system can perform memory-to-memory operations, leaving the registers free.

Through the PDP-11's auto-increment and auto-decrement modes, pointer manipulation is easy, simplifying the handling of such structured data as arrays and character strings. To step through a table in the forward direction, the pointer is auto-incremented; when the decrement mode is used, the same pointer can be backed up.

HARDWARE STACKS

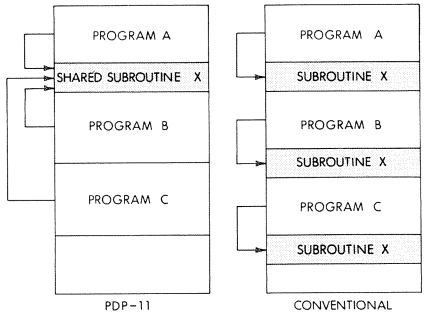
For convenient handling of frequently used data, PDP-11 computers employ a hardware pushdown stack, a powerful built-in data storage feature normally found only on larger computers.

Stacks speed the servicing of interrupts and subroutine calls by providing fast temporary storage of program information. For example, when an interrupt occurs, the status of the interrupted routine is stored or "nested" on the stack. As soon as the higher priority task has been completed, the status information is "popped off" of the stack and the interrupted routine is resumed automatically.

Without the stack, much core and time would be wasted, since a separate subroutine would be required to store the status information on each interrupted task.

With stacks, users can also share common (reentrant) code, saving much valuable core space. Instead of each user requiring his own copy of a particular routine, several users can share the same routine copy.

A user program, for example, can start executing the common code and be interrupted by another program which also requires the code. The stack stores information on the breakpoints so that each user program can reenter the code at the proper point and finish execution.



DIRECT MEMORY ACCESS

PDP-11

Standard on all PDP-11's. Transfer rate: 5 million bytes per second.

CONVENTIONAL

Sometimes available as a costly option. Usually requires a separate DMA channel for each device.

HARDWARE INTERRUPTS

The interrupt system for the PDP-11 is another new departure in small computer technology. With fully vectored interrupts, the system eliminates the high overhead software that determines which device service routine to use and the code necessary to save system status. In addition, the multi-level hardware interrupt system is a standard PDP-11 feature not an extra-cost option.

The PDP-11 system consists of four priority levels, each of which can handle an almost unlimited number of devices. The priority of the device is a function of the device's physical position—the closer to the processor, the higher its priority on that level.

The priority system makes an excellent use of the PDP-11's hardware stacks. When the processor services an interrupt, it first saves important program information on the stack. This information enables the processor to automatically return to the same point in the program and the same conditions, once the current interrupt or interrupts have been serviced.

In the PDP-11, the device causing the interrupts provides a direct vector to its own service routine, eliminating the slow and tedious operation of polling all devices to see which one interrupted.

The device also provides status information for its own service routine. Thus the programmer has the flexibility of assigning a device to a higher priority and its service routine to a lower priority without writing special software.

The system also allows interrupts to be enabled or disabled, through software, during program operation. Such masking allows priorities to change dynamically in response to system conditions. For example, a real-time program could disable data entry terminals whenever critical analog data is being collected. As soon as the scan was complete, the terminals would be automatically enabled and be ready to input data.

With the PDP-11, any number of interrupts can be enabled or disabled; other systems are restricted to 16 by virtue of their word length.

REAL-TIME CLOCK

For accuracy in timing intervals or counting events, the PDP-11/10 includes a real-time clock, another standard feature. For time dependent applications, the clock provides interrupts or flags every 16 milliseconds for a line frequency of 60 Hz, or every 20 milliseconds for a line frequency of 50 Hz.

POWER FAIL

PDP-11	CONVENTIONAL
Standard. Guarantees program and equipment protection and automatic restart.	Usually extra cost option. Typically guarantees memory contents, nothing else.

POWER FAIL PROTECTION AND AUTOMATIC RESTART

Power fail protection and automatic restart are standard PDP-11 features which protect user programs and minimize downtime in the event of power outages or fluctuations.

When the system senses a failure or severe power fluctuation, it has a sufficient amount of stored power from the power supply to protect the program in memory and shut down the system in orderly fashion. A user-developed shutdown program protects system and special devices from harm and saves important registers.

When the power returns to safe operating levels, the system is automatically restarted via a user programmed routine. Unattended systems can thus resume operation without human intervention.

INSTRUCTION SET

Single Operand		
CLR	(B)	Clear
COM	(B)	Complement
NEG	(B)	Negate
INC	(B)	Increment
DEC	(B)	Decrement
ADC	(B)	Add Carry
SBC	(B)	Subtract Carry
TST	(B)	Test
ROR	(B)	Rotate Right
ROL	(B)	Rotate Left
ASR	(B)	Arithmetic Shift Right
ASL	(B)	Arithmetic Shift Left
SWAB		Swap Bytes
Branch		
BR		Unconditional Branch
BEQ		Equal (Zero)
BNE		Not Equal
BMI		Minus
BPL		Plus
BCS/C		Carry Set/Clear
BVS/C		Overflow Set/Clear
BLT		Less Than
BGE	Signed Integer	Greater Than or Equal
BLE	Signed Integer	Less Than or Equal
BGT	Signed Integer	Greater Than
BHI	Unsigned Integer	Higher Than
BLOS	Unsigned Integer	Lower or Same
BHIS	Unsigned Integer	Higher or Same
BLO	Unsigned Integer	Lower Than

Double Operand		
MOV	(B)	Move
ADD		Add
SUB		Subtract
CMP	(B)	Compare
BIT	(B) Boolean	Bit Test
BIC	(B) Boolean	Bit Clear
BIS	(B) Boolean	Bit Set
Control		
JMP		Jump
JSR		Subroutine Jump
RTS		Return From Subroutine
RTI		Return From Interrupt
Condition Code		
SET	C, V, N, Z	Set Selected Bit(s)
CLR	C, V, N, Z	Clear Selected Bit(s)
Miscellaneous		
HALT		Stop
WAIT		Wait for Interrupt
RESET		Initialize
Trap		
IOT		I/O Call
EMT		Emulator Call
TRAP		User Call
BPT		Breakpoint

(B)=Byte Instruction.

HARDWARE STACKS

PDP-11

Standard hardware features on all models. Provides automatic temporary data storage. Improves efficiency, reduces programming costs.

CONVENTIONAL

Not normally available. Must be implemented by software at the expense of programming time, core memory space, and execution speed.

PDP-11 SOFTWARE

PDP-11 software ranges from broad-based monitors to specialized applications packages.

The PDP-11's Disk Operation System provides FORTRAN IV that meets ANSI standards. And the system's paper tape software includes development and utility programs and provides both single and eight-user BASIC systems.

An industrial control monitor and a communications monitor are available via DIGITAL's Industrial Products and Communications Groups, respectively.

COMTEX-11 is employed in message switching, remote terminal systems, line concentrators, front ends, etc. Modular and easily adapted to special requirements, COMTEX-11 is the basis for the DECcomm system software packages.

RSX-11, the PDP-11 industrial control software package, is designed to coordinate the execution of tasks in a multi-programming environment. It provides scheduling, input/output, operator communication, and other related functions.

PDP-11 DISK OPERATING SYSTEM (DOS)

The PDP-11 Disk Operating System is a core and disk resident software system which provides the user with the programs he needs for efficient program development and on-line or batch mode execution. To control the DOS monitor and its subsystems, the user issues simple commands via the system's console teleprinter.

For program development, DOS provides a relocatable assembler (MACRO-11), a FORTRAN IV compiler that meets full ANSI standards, on-line editing and debugging programs, and a file utilities package (PIP). For simplified program execution, DOS provides common I/O device handling routines, a linking loader (LINK-11), and operator interface software.

MACRO-11 is a powerful assembler that provides the user with full macro capability, complete or partial listing of the symbolic program, and lists of symbols with cross reference. Code generated by MACRO-11 is relocatable. Therefore programs can be assembled in separate, easy-to-handle modules and then linked for loading.

LINK-11 combines the separately assembled modules and/or FORTRAN compilations into a single load module. The loader also allows the user to overlay modules stored on disk on top of program segments in core. Overlaying makes it possible to execute programs in segments so that a total program can be larger than the available core memory.

The user can also perform trade-offs to optimize for either processing speed or available core space. By making most modules core resident, he can improve processing speed. Conversely, if he keeps most modules on disk and swaps only when they are required, he can have more space available for other requirements.

The system provides for both sequential and random access file handling. The size of sequential files on disk or DECtape need not be specified in advance; they may grow or shrink dynamically as they are processed. Random access processing uses a powerful directory to permit efficient file access.

The PDP-11 file protection system allows a user to choose from a variety of file protection levels. For example, he can restrict a certain group of users to read-only access of a particular file or protect a file against his own inadvertent deletion. Protection is accomplished through a software identification code.

With DOS software, I/O devices are buffered, so that I/O operations may be overlapped with processing for maximum hardware utilization. Programs are device independent; that is, the user has until run time to specify a device and can respecify the device at any time.

BYTE HANDLING

PDP-11	CONVENTIONAL
Can operate on bits, bytes, words or multiwords. 17 basic bytes instructions. All instructions operate on registers, memory locations or peripheral device registers.	Usually operates on single words—limited or no byte handling capability. Arithmetic instructions usually operate on accumulators only.

FORTRAN IV

FORTRAN IV operates under PDP-11 DOS, using DOS monitor I/O calls and all DOS peripherals.

PDP-11 FORTRAN provides many language extensions: random access I/O, mixed mode arithmetic, generalized expressions used as array subscripts. An IMPLICIT statement allows the user to conveniently control variable type.

Error diagnostics are improved through an error traceback feature that specifies when the error occurred and all the linkages back to the main program. PDP-11 FORTRAN also allows the user to define how many times errors (e.g., arithmetic overflow) can occur before they become fatal.

Arithmetic can be performed with or without the PDP-11 extended arithmetic element; PDP-11 FORTRAN will provide up to 24-bit accuracy for two-word (real) formats or up to 56-bit accuracy for four-word (double precision) formats.

PAPER TAPE SOFTWARE

Paper tape software for the PDP-11 lets the user develop programs in PAL-11 assembly language or in single-user or 8-user BASIC.

The executive consists of device driver packages and the input/output routines for such devices as the teletypewriter, line printer, and high speed reader/punch.

Paper tape software also includes an on-line editor (ED-11), absolute and linking loaders, an on-line debugging program (ODT-11), and a floating point and math package (FPP-11). The latter package provides the user with common mathematical subroutines which are reentrant or shareable for maximum utilization.

Single-User BASIC

BASIC is the popular problem-solving interactive language that was developed at Dartmouth College and is widely used for scientific and other applications. The language is simple enough to be easily learned by a beginner, yet provides features which give the sophisticated programmer wide flexibility.

With single-user BASIC, the PDP-11 can be used as a calculator, since commands can be executed immediately or stored for later execution. Also, BASIC programs written on other systems can run on the PDP-11 with very few or no modifications.

Single-user BASIC allows machine language subroutines to be part of any BASIC program—a feature which is extremely valuable in data acquisition applications. For example, a major part of a particular program can be written in BASIC language and input/output routines can be written in machine language. Another feature important in program development is error reporting; errors detected by the system are automatically output on the teletypewriter or terminal as development proceeds.

8-User BASIC

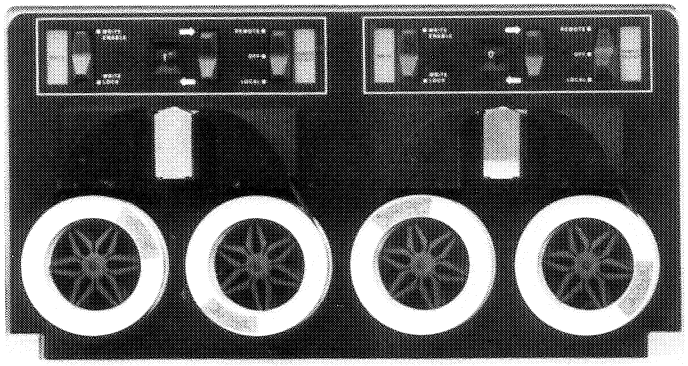
Eight-user BASIC is an extension of single-user BASIC which allows up to 8 users simultaneous access to the PDP-11.

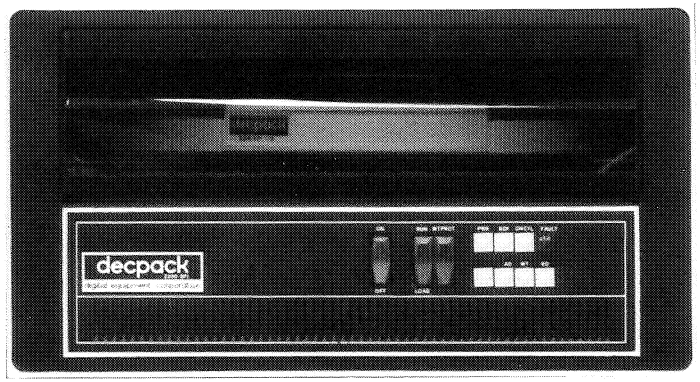
In addition to the advantages of single-user BASIC, eight-user BASIC provides:

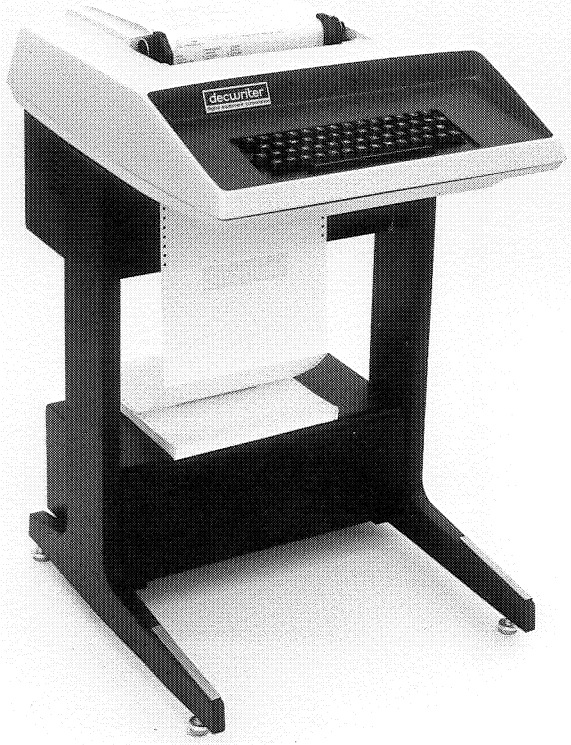
- Buffered interrupt-driven I/O
- "Open" and "Close" commands which allow BASIC users to select a particular peripheral device such as the high speed paper tape reader/punch or line printer
- Modified "Input" and "Print" commands to let the user program communicate with the device.
- Additional print functions, including character, space, and tab
- Improved diagnostics

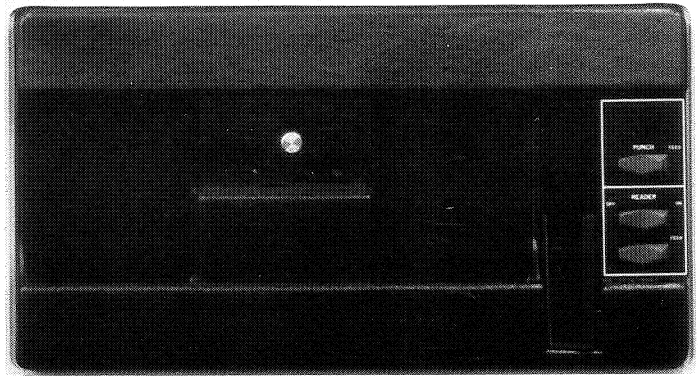
I/O INSTRUCTIONS

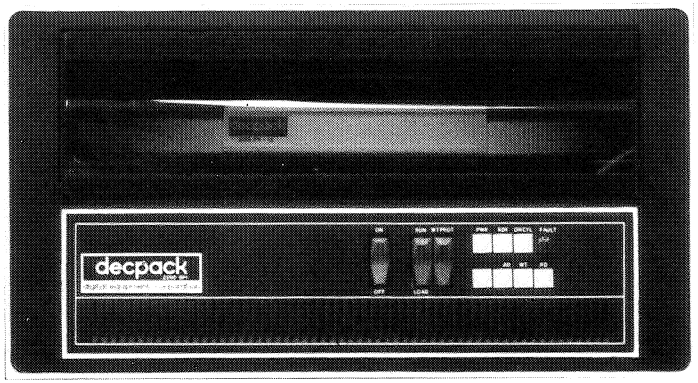
PDP-11	CONVENTIONAL
No I/O instructions needed. All instructions operate on device registers. Simplifies programming.	Separate I/O instructions required for data transfers. Complicates and limits programming.

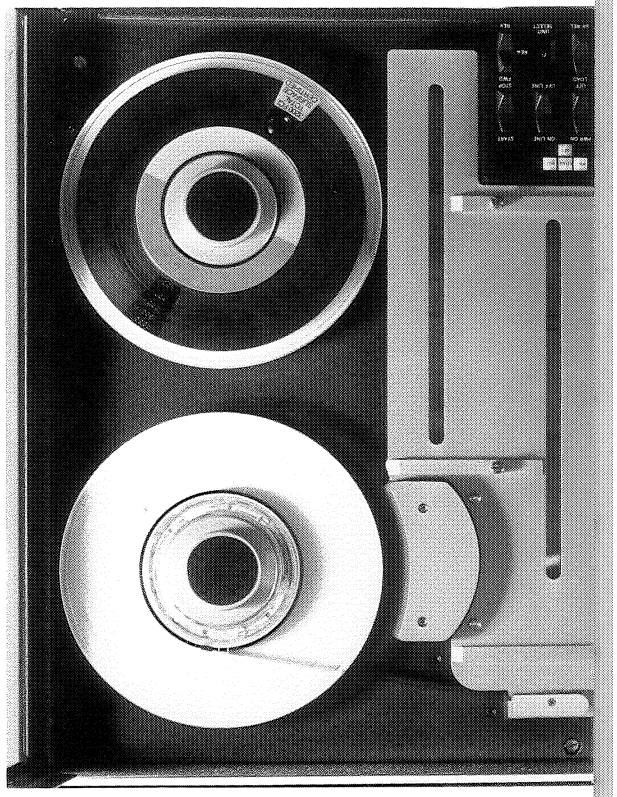












REGISTERS

PDP-11

Eight general purpose registers. No special accumulators required. Registers may be used as accumulators, index registers or pointers.

CONVENTIONAL

Usually one or two index registers or one or two accumulators. Complicated register/accumulator schemes.

PERIPHERALS

ALPHANUMERIC KEYBOARD DISPLAY (VT05)
Low cost, high performance, alphanumeric keyboard display. Provides seven selectable speeds from 110 to 2400 baud. Meets EIA RS232C standards and 20 mA current loop requirements. 64 or 128 ASCII character sets.

DECWRITER DATA TERMINAL (LA30)
Fast, low cost DIGITAL-designed terminal. Prints asynchronously at 30 characters per second. 64 character print set; 96 or 128 character keyboard. Quieter than an electric typewriter. Simple mechanical design for high reliability.

TELETYPES (LT33)
ASR 33 Teletype. Reads, prints, and punches at 10 characters per second. Other models available.

PAPER TAPE READER/PUNCH (PC11)
Reader operates at 300 characters per second. Punch operates at 50 characters per second.

CARD READERS (CR11, CM11, CD11)
Reads up to 1000 cards per minute. Available in mark-sense, punched card and DMA punched card versions.

HIGH SPEED LINE PRINTER (LP11)
Produces up to 1200 lines per minute. 80 or 132 columns. 64 or 96 character print sets.

DECTAPE (TU56)
DIGITAL's economical and unique variety of magnetic tape. Easy to use, transport and store. Convenient 3.9 inch diameter reels hold up to 131,072 16-bit words each and may be updated in blocks. Dual drive system. Insensitive to line voltage or frequency variations. High reliability due to simple design. Ideal for production environment.

DEC MAGNETIC TAPE (TU10)
Industry compatible magnetic tape, 7 or 9 track. Transfers up to 36,000 characters per second. High density—180 million bits on 9-track and 135 million bits on 7-track. Up to 8 TU10 transports per TM11 controller. DIGITAL-designed unit provides character-by-character parity checks, automatic longitudinal redundancy check, and automatic cycle redundancy check for 9-track unit.

DECPACK DISK CARTRIDGE SYSTEM (RK05)
Economical, large volume removable disk storage. 1.2 million words per drive; 11.08 μsec/word transfer rate. 70 msec average access time. 8 independent drivers per controller for a total system capacity of 9.6 million words.

DECDISK SYSTEM (RS64)
Fast, low cost DIGITAL-designed fixed head disk system. A single DECdisk and control provides 64 K words of storage. RC11 controller will operate up to 4 disks for a total of 262,144 words of storage. Full cycle redundancy data checking. Rugged packaging for industrial applications.

DISK AND CONTROL (RF11/RS11)
Fast, fixed head disk system. 16 μsec per word transfer rate. 17 msec average access time. RS11 drive capacity is 256 K. RF11 controller can operate up to 8 drives for a total capacity of over 2 million words.

STORAGE DISPLAY (VT01)
Tektronix Type 611 direct view storage scope. Resolution: 300 stored line pairs vertically and 300 line pairs horizontally. Displays 4000 flicker free characters or 30,000 discrete resolvable points.

POINT PLOT DISPLAY (VR14)
Compact CRT display with 6-3/4 x 9 inch view area (19 inch package). Provides bright point plot displays.

HARDWARE INTERRUPT SYSTEM

PDP-11
Four levels allow multiple devices on each line.

CONVENTIONAL
Usually one level only with limited number of devices.

COMMUNICATIONS EQUIPMENT

ASYNCHRONOUS SINGLE LINE INTERFACES (DL11)

Connects PDP-11 systems to a variety of communication channels. Features include double character-buffered receiver and transmitter, selectable data rates (between 50 and 9600 baud), independent receive and transmit speeds, strap selectable character size and stop code length, 20 mA or EIA output levels, and optional full dataset control.

PROGRAMMABLE ASYNCHRONOUS DUAL LINE INTERFACE (DC11)

Interfaces local or remote terminals (via modems or datasets) to PDP-11 systems. Full or half duplex operation at 4 programmable line speeds. Split speed operation. Programmable character size: 5, 6, 7, or 8 bits. Automatic parity checking. Auto-answering capability. Interfaces to Bell 103, 202 or equivalent datasets. Reverse channel available for Bell 202 operation.

ASYNCHRONOUS 16-LINE INTERFACE (DM11)

Interfaces local or remote terminals (via modems or datasets) to PDP-11's. Up to 16 DM11's per PDP-11 for a total of up to 256 full duplex lines. Full or half duplex operation at rates up to 1200 baud. Characters assembled in, and messages transmitted from, core memory (DMA). Character size is jumper-selectable: 5, 6, 7, or 8 bits. Incoming characters receive automatic parity check, break detection, reverse break generation and are buffered in a 64 character "tumble" table. Modem control and various line drivers are optional.

SYNCHRONOUS LINE INTERFACE (DP11)

Interfaces high speed local or remote terminals or other computers to PDP-11's. Provides double buffering, full or half duplex operation, and programmable "sync" character and "sync" character stripping. Character size is programmable: 6, 7, or 8 bits. Auto-answering capability. Interfaces to Bell 201 and 303 or equivalent datasets. Internal clock is optional.

AUTOMATIC CALLING UNIT INTERFACE (DN11)

Provides computer control of Bell 801A, 801C or equivalent Automatic Calling Units.

SIGNAL CONDITIONING INTERFACES (DF11)

DF11 signal conditioning options permit most DIGITAL serial line interfaces to adapt to any of the common communications levels such as 20 mA teletype or EIA. The options include both the electrical conversion circuitry and the physical electrical connectors. One DF11 model incorporates an internal modem so that dataset requirements can be bypassed.

COMMUNICATIONS ARITHMETIC OPTION (KG11-A)

Computes cyclic redundancy checks (CRC) and longitudinal redundancy checks (LRC) for detecting errors in serially transmitted data.

INDUSTRIAL INTERFACES

ANALOG SCANNER SYSTEM (AFC11)

A true industrial subsystem for low level differential analog inputs. Expands to 1024 channels. High noise rejection.

UNIVERSAL DIGITAL CONTROLLER (UDC11)

Handles discrete process input/output such as contacts, relays, switches, pushbuttons, drivers for lamps or solenoids, counters and analog outputs. Expands to 4096 points.

ANALOG TO DIGITAL CONVERSION SUBSYSTEM (AD01-D)

Handles single-ended high level analog inputs. Optional bipolar with automatic sign. 10-bit precision. 14-bit resolution.

DIGITAL TO ANALOG CONVERTER (AA11-D)

Handles analog outputs. 11-bit precision plus sign. Bipolar output.

VECTORED INTERRUPTS

PDP-11

Standard feature. Each device provides a direct vector to its own service routine. Automatically handled by hardware.

CONVENTIONAL

Software polling of I/O devices usually required.

MEMORY

PDP-11

Memories of various speeds may be intermixed due to the system's asynchronous bus.

CONVENTIONAL

Machines that operate synchronously are restricted to a single memory speed.

PDP-11 FAMILY SUMMARY

PDP-11/05 and PDP-11/15 let the original equipment manufacturer (OEM) tailor a machine to specific requirements. With the PDP-11/15, he can select consoles, specify the type and amount of memory, and choose from a variety of options such as power/fail restart, and single or multi-line interrupt capability. These options are standard on the PDP-11/05. Both machines are comparable to the PDP-11/20 in computational power and are UNIBUS and binary compatible with all PDP-11 family members.

PDP-11/20 is one of the most widely accepted computers on the market today. With its flexible instruction set and word and byte addressing, the system is ideal for communications applications such as data concentration and message switching. It is also used in physics, biomedicine, education, industry, business, computation and research, serving in such widely diverse applications as order entry and spectroscopy.

PDP-11R20 is designed specifically for use in severe environments...to operate in moving vehicles, on ships and planes, and in environments where shock, vibration, motion and other factors might cause a less rugged computer to fail. It has all PDP-11/20 features plus a welded chassis, heavy duty fans, sealed switches, drip-proof construction and special clamps and reinforcing bars. The system's front panel is constructed of metal and is removable so that it can be operated remotely. The design of EMI protected throughout. Reports verifying conformance to various military specifications are available on request.

Operating Shock	5G, 11 msec	3 shocks in each direction on 3 mutually perpendicular axes (18 shocks).
Non-operating shock	15G, 11 msec	
Vibration	5-9 Hz, 1.0" double amplitude 9-500 Hz, 2.5G	vibration applied on 3 mutually perpendicular axes
Power Line Frequency	47 to 420 Hz	
Power Line Voltage	106, 118 VAC ±10% 212, 224, 236 VAC ±10%	
Operating Temperature	0° C to + 55° C	
Non-operating Temperature	−55° C to + 85° C	

PDP-11/40 is the anchor member of the PDP-11 family, providing all the features a user may need to achieve large expansion, with a reasonable first investment. With memory management, the user can expand to 128 K of addressable memory and protect user programs in core. The system also includes an expanded instruction set and a hardware floating point processor.

PDP-11/45, the most powerful PDP-11 family member, is an excellent computation tool for large multi-user multi-task installations. Through memory management, memory can expand to 128 K which can include a combination of bipolar and MOS memory. MOS and bipolar memories are dual-ported so that computation can be overlapped for fast throughput. Other computation features include a greatly expanded floating point processor.

PERIPHERALS

PDP-11	CONVENTIONAL
DIGITAL is a single source for processors, memory, peripherals. More choices, better reliability, quantity discounts.	Most companies manufacture mainframes only. Choice of peripherals is frequently limited.