What Have We Learned from the PDP–11?

C. Gordon Bell

Digital Equipment Corporation
Maynard, Massachusetts
and
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania

In the six years that the PDP–11 has been on the market, more than 20,000 units in 10 different models have been sold. Although one of the original system design goals was a broad range of models, the actual range of 500 to 1 (in cost and memory size) has exceeded the design goals.

The PDP–11 was designed to be a small computer, yet its design has been successfully extended to high-performance models. This paper recollects the experience of designing the PDP–11, commenting on its success from the point of view of its goals, its use of technology, and on the people who designed, built and marketed it.

1. INTRODUCTION

A computer is not solely determined by its architecture; it reflects the technological, economic, and human aspects of the environment in which it was designed and built. Most of the non-architectural design factors lie outside the control of the designer: the availability and price of the basic electronic technology, the various government and industry rules and standards, the current and future market conditions. The finished computer is a product of the total design environment.

In this chapter, we reflect on the PDP–11: its goals, its architecture, its various implementations, and the people who designed it. We examine the design, beginning with the architectural specifications, and observe how it was affected by technology, by the development organization, the sales, application, and manufacturing organizations, and the nature of the final users. Figure 1 shows the various factors affecting the design of a computer. The lines indicate the primary flow of information for product behavior and specifications. The physical flow of materials is along nearly

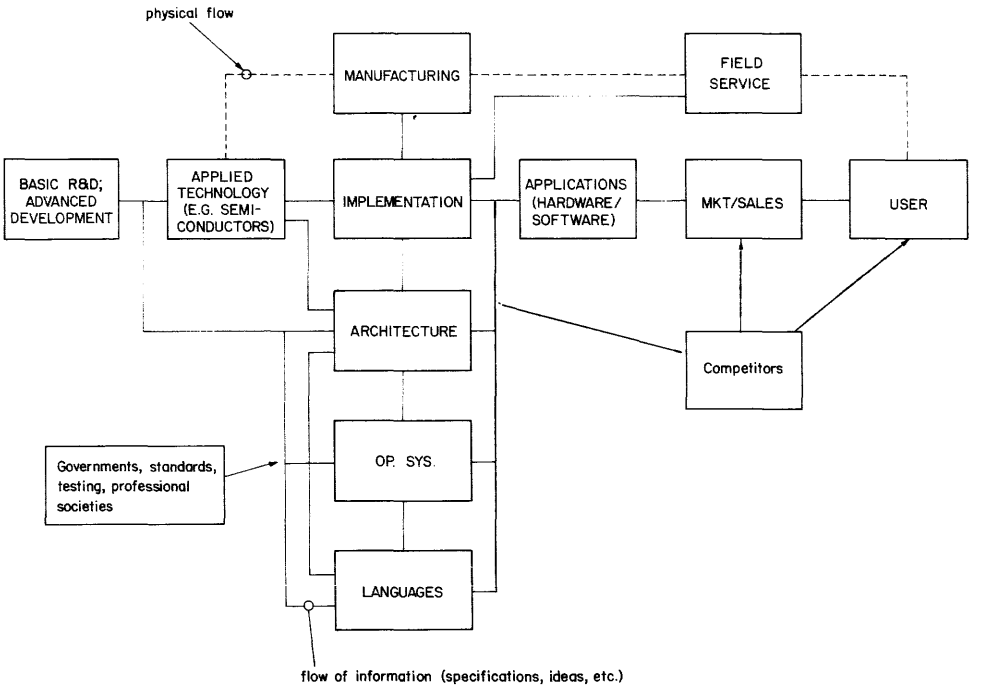


Fig. 1. Structure of organization affecting a computer design.

the same lines, but more direct: beginning with the applied technology manufacturers, material moves through computer manufacturing and then to service personnel before delivery to the end user.

2. BACKGROUND: THOUGHTS BEHIND THE DESIGN

It is the nature of computer engineering to be goal-oriented, with pressure to produce deliverable products. It is therefore difficult to plan for an extensive lifetime. Nevertheless, the PDP–11 evolved rapidly, and over a much wider range than we expected. This rapid evolution would have placed unusual stress even on a carefully planned system. The PDP–11 was not extremely well planned or controlled; rather it evolved under pressure from implementation and marketing groups.

Because of the many pressures on the design, the planning was asynchronous and diffuse; development was distributed throughout the company. This sort of decentralized design organization provides a system of checks and balances, but often at the expense of perfect hardware compatibility. This compatibility can hopefully be provided in the software, and at lower cost to the user.

Despite its evolutionary planning, the PDP–11 has been quite successful in the marketplace: over 20,000 have been sold in the six years that it has been on the market (1970–1975). It is not clear how rigorous a test (aside from the marketplace) we have given the design, since a large and aggressive marketing organization, armed with software to correct architectural inconsistencies and omissions, can save almost any design.

It has been interesting to watch as ideas from the PDP–11 migrate to other computers in newer designs. Although some of the features of the PDP–11 are patented, machines have been made with similar bus and ISP structures. One company has manufactured a machine said to be "plug compatible" with a PDP–11/40. Many designers have adopted the UNIBUS as their fundamental architectural component. Many microprocessor designs incorporate the UNIBUS notion of mapping I/O and control registers into the memory address space, eliminating the need for I/O instructions without complicating the I/O control logic. When the LSI–11 was being designed, no alternative to the UNIBUS-style architecture was even considered.

An earlier paper [Bell *et al.* 70] described the design goals and constraints for the PDP–11, beginning with a discussion of the weaknesses frequently found in minicomputers. The designers of the PDP–11 faced each of these known minicomputer weaknesses, and our goals included a solution to each one. In this section we shall review the original design

goals and constraints, commenting on the success or failure of the PDP–11 at meeting each of them.

The first weakness of minicomputers was their limited addressing capability. The biggest (and most common) mistake that can be made in a computer design is that of not providing enough address bits for memory addressing and management. The PDP–11 followed this hallowed tradition of skimping on address bits, but it was saved by the principle that a good design can evolve through at least one major change.

For the PDP–11, the limited-address problem was solved for the short run, but not with enough finesse to support a large family of minicomputers. That was indeed a costly oversight, resulting in both redundant development and lost sales. It is extremely embarassing that the PDP–11 had to be redesigned with memory management only two years after writing the paper that outlined the goal of providing increased address space. All predecessor DEC designs have suffered the same problem, and only the PDP–10 evolved over a long period (ten years) before a change was needed to increase its address space. In retrospect, it is clear that since memory prices decline 26 to 41% yearly, and users tend to buy "constant-dollar" systems, then every two or three years another address bit will be required.

A second weakness of minicomputers was their tendency not to have enough registers. This was corrected for the PDP–11 by providing eight 16-bit registers. Later, six 32-bit registers were added for floating-point arithmetic. This number seems to be adequate: there are enough registers to allocate two or three (beyond those already dedicated to program counter and stack pointer) for program global purposes and still have registers for local statement computation. More registers would increase the multiprogramming context switch time and confuse the user.

A third weakness of minicomputers was their lack of hardware stack capability. In the PDP–11, this was solved with the autoincrement/autodecrement addressing mechanism. This solution is unique to the PDP–11 and has proven to be exceptionally useful. (In fact, it has been copied by other designers.)

A fourth weakness, limited interrupt capability and slow context switching, was essentially solved with the device of UNIBUS interrupt vectors, which direct device interrupts. Implementations could go further by providing automatic context saving in memory or in special registers. This detail was not specified in the architecture, nor has it evolved from any of the implementations to date. The basic mechanism is very fast, requiring only four memory cycles from the time an interrupt request is issued until the first instruction of the interrupt routine begins execution.

A fifth weakness of prior minicomputers, inadequate character-handling capability, was met in the PDP–11 by providing direct byte addressing

capability. Although string instructions are not yet provided in the hardware, the common string operations (move, compare, concatenate) can be programmed with very short loops. Benchmarks have shown that systems which depend on this string-handling mechanism do not suffer for it.

A sixth weakness, the inability to use read-only memories, was avoided in the PDP–11. Most code written for the PDP–11 tends to be pure and reentrant without special effort by the programmer, allowing a read-only memory (ROM) to be used directly. ROMs are used extensively for bootstrap loaders, program debuggers, and for normal simple functions. Because large ROMs were not available at the time of the original design, there are no architectural components designed specifically with large ROMs in mind.

A seventh weakness, one common to many minicomputers, was primitive I/O capabilities. The PDP–11 answers this to a certain extent with its improved interrupt structure, but the more general solution of I/O processors has not yet been implemented. The I/O-processor concept is used extensively in the GT4X display series, and for signal processing. Having a single machine instruction that would transmit a block of data at the interrupt level would decrease the CPU overhead per character by a factor of three, and perhaps should have been added to the PDP–11 instruction set.

Another common minicomputer weakness was the lack of system range. If a user had a system running on a minicomputer and wanted to expand it or produce a cheaper turnkey version, he frequently had no recourse, since there were often no larger and smaller models with the same architecture. The problem of range and how it is handled in the PDP–11 is discussed extensively in a later section.

A ninth weakness of minicomputers was the high cost of programming them. Many users program in assembly language, without the comfortable environment of editors, file systems, and debuggers available on bigger systems. The PDP–11 does not seem to have overcome this weakness, although it appears that more complex systems are being built successfully with the PDP–11 than with its predecessors, the PDP–8 and PDP–15. Some systems programming is done using higher-level languages; the optimizing compiler for BLISS–11, however, runs only on the PDP–10.

One design constraint that turned out to be expensive, but probably worth it in the long run, was that the word length had to be a multiple of eight bits. Previous DEC designs were oriented toward 6-bit characters, and DEC has a large investment in 12-, 18-, and 36-bit systems. The notion of word length is somewhat meaningless in machines like the PDP–11 and the IBM System/360, because data types are of varying length, and instructions tend to be multiples of 16 bits.

Microprogrammability was not an explicit design goal, partially since

the large ROMs which make it feasible were not available at the time of the original Model 20 implementation. All subsequent machines have been microprogrammed, but with some difficulty and expense.

Understandability as a design goal seems to have been minimized. The PDP–11 was initially a hard machine to understand, and was marketable only to those who really understood computers. Most of the first machines were sold to knowledgeable users in universities and research laboratories. The first programmers' handbook was not very helpful, and the second, arriving in 1972, helped only to a limited extent. It is still not clear whether a user with no previous computer experience can figure out how to use the machine from the information in the handbooks. Fortunately, several computer science textbooks [Gear 74, Eckhouse 75, and Stone and Siewiorek 75] have been written based on the PDP–11; their existence should assist the learning process.

We do not have a very good understanding of the style of programming our users have adopted. Since the machine can be used so many ways, there have been many programming styles. Former PDP–8 users adopt a one-accumulator convention; novices use the two-address form; some compilers use it as a stack machine; probably most of the time it is used as a memory-to-register machine with a stack for procedure calling.

Structural flexibility (modularity) was an important goal. This succeeded beyond expectations, and is discussed extensively in the UNIBUS section.

3. TECHNOLOGY: COMPONENTS OF THE DESIGN

The nature of the computers that we build is strongly influenced by the basic electronic technology of their components. The influence is so strong, in fact, that the four generations of computers have been named after the technology of their components: vacuum-tube, single semiconductors (transistors), integrated circuits (multiple transistors packaged together), and LSI (large-scale integration). The technology of an implementation is the major determinant of its cost and performance, and therefore of its product life. In this section, we shall examine the relationship of computer design to its supporting technology.

3.1. Designing with Improved Technologies

Every electronic technology has its own set of characteristics, all of which the designer must balance in his choice of a technology. They have different cost, speed, heat dissipation, packing density, reliability, etc. These factors combine to limit the applicability of any one technology; typically, we use one until we reach some such limit, then convert to

another. Often the reasons for the existence of a newer technology will lie outside the computer industry: integrated circuits, for example, were developed to meet aerospace requirements.

When an improved basic technology becomes available to a computer designer, there are three paths he can take to incorporate that technology in a design: use the newer technology to build a cheaper system with the same performance, hold the price constant and use the technological improvement to get a slight increase in performance, or push the design to the limits of the new technology, thereby increasing both performance and price.

If we hold the cost constant and use the improved technology to get better performance, we will get the best increase in total-system cost effectiveness. This approach provides a growth in quality at a constant price and is probably the best for the majority of existing users.

If we hold the performance constant and use the improved technology to lower the cost, we will build machines that make new applications possible. The minicomputer (for *mini*mal *computer*) has traditionally been the vehicle for entering new applications, since it has been the smallest computer that could be constructed with a given technology. Each year, as the price of the minimal computer continues to decline, new applications become economically feasible. The microprocessor, or processor-on-a-chip, is a new yet evolutionary technology that again provides alternatives to solve new problems.

If we use the new technology to build the most powerful machine possible, then our designs advance the state of the art. We can build completely new designs that can solve problems previously not considered feasible. There are usually two motivations for operating ahead of the state of the art: preliminary research motivated by the knowledge that the technology will catch up, and national defense, where an essentially infinite amount of money is available because the benefit—avoiding annihilation—is infinite.

3.2. Specific Improvements in Technology

The evolution of a computer system design is shaped by the technology of its constituent parts. Machines of varying sizes will often be based on different technologies, and will therefore evolve differently as the component evolutions differ.

The evolutionary rate of computers has been determined almost exclusively by the technology of magnetic storage and semiconductor logic. Memory is the most basic component of a computer, and it is used throughout the design. Besides the obvious uses as "main" program memory

and file storage devices (disks and tapes), we find memory inside the processor in the form of registers and indicators, memory as a cache between the processor and the program memory, and memory in I/O devices as buffers and staging areas. Memory can be substituted for nearly all logic, by substituting table-lookup for computation. We are therefore deeply interested in all forms of memory used throughout a computer system.

Disk technology has evolved in a different style than the primary memory devices. The price of a physical structure, such as a disk pack with 10 platters, actually increases somewhat in time, but the storage density per square inch increases dramatically. IBM's disk packs with 10 platters (beginning with the 1311) have increased in capacity at the rate of 42% per year, and the price per bit has decreased at 38% per year. For disks, there has been an economy of scale; the number of bits for larger disk units increases more rapidly than cost, giving a decreased cost per bit. The overhead of motors, power supply, and packaging increases less rapidly with size. Also, more effort is placed to increasing the recording density of larger disks. The technology developed for the large disks shows up on smaller designs after several years.

The price of a chip of semiconductor memory is essentially independent of its size. A given design is available for one or two years at a high price (about $10 per chip) while people buy in to evaluate the component. Within a year, enough have been sold that the price drops sharply, sometimes by more than a factor of two. The density of state-of-the-art laboratory semiconductor memory has doubled every year since 1962. The relative growth of different semiconductor technologies is shown in Table I; the density of MOS read/write is used as a reference.

Keeping in mind this simple model for the growth of specific semiconductor technologies, we find the situation shown in Table II.

The various technology improvement rates per year for other major components are 30% for magnetic cores, 25% for terminals, 23% for magnetic tape density, 29% for magnetic tape performance, and –3% for packaging and power. The total effect of all component technologies on minicomputers has been an average price decline of about 31% per year

TABLE I

Bipolar read/write	Lags by 2 years
Bipolar read-only	Lags by 1 year
MOS read/write	(Reference year)
MOS read-only	Leads by 1 year
Production volumes	Lags by 1 or 2 years

TABLE II

Technology	Bits	Production availability
Bipolar read/write	16	1969–1970
	64	1971–1972
	1024	1975–1976
MOS read/write	16,384	1977–1978
Bipolar read-only	256	1971–1972
	1024	1974–1975
	2048	1975–1976

[Bell and Newell 71]. In 1972 an 8-bit 1-chip microprocessor was introduced; the price of these machines is declining at a rate comparable to the 12- and 16-bit minicomputers. In summary, if we look at the price/performance behavior of computers over time, the computer performance simply tracks memory performance.

3.3. PDP-11 Evolution through Memory Technologies

The design of the PDP–11 series began in 1969 with the Model 20. Subsequently, three models were introduced as minimum-cost, best cost/performance, and maximum-performance machines. The memory technology of 1969 imposed several constraints. First, core memory was cost effective for the primary (program) memory, but a clear trend toward semiconductor primary memory was visible. Second, since the largest high-speed read/write memories available were 16 words, then the number of processor registers should be kept small. Third, there were no large high-speed read-only memories that would have permitted a microprogrammed approach to the processor design.

These constraints established four design attitudes toward the PDP–11's architecture. First, it should be asynchronous, and thereby capable of accepting different configurations of memory that operate at different speeds. Second, it should be expandable to take eventual advantage of a larger number of registers, both user registers for new data types and internal registers for improved context switching, memory mapping and protected multiprogramming. Third, it could be relatively complex, so that a microcode approach could eventually be used to advantage: new data types could be added to the instruction set to increase performance, even though they might add complexity. Fourth, the UNIBUS width should be relatively large, to get as much performance as possible, since the amount of computation possible per memory cycle is relatively small.

As semiconductor memory of varying price and performance became available, it was used to trade cost for performance across a reasonably wide range of models. Different techniques were used on different models to provide the range. These techniques include microprogramming to enhance performance (for example, faster floating point), use of faster program memories for brute-force speed improvements, use of fast caches to optimize program memory references, and expanded use of fast registers inside the processor.

3.3.1. MICROPROGRAMMING

Microprogramming is the technique by which the conventional logic of a sequential machine is replaced by an encoded representation of the logic and circuitry to decode that microprogram. Microprograms are stored in conventional random-access memories, though often in read-only versions.

Computer designers use microprogramming because it permits relatively complex control sequences to be generated without proportionately complex logic. It is therefore possible to build more complex processors without the attendant problems of making such large sequential circuits. Additionally, it is much easier to include self-diagnosis and maintenance features in microprogrammed logic. Microprogramming depends on the existence of fast memories to store the microprograms. The memory speed is determined by the task at hand: whether it be for an electric typewriter or a disk controller, the microprogram memory must be fast enough to keep up with the application. Typically, processors and fast disks present the most difficult control problems.

Microprogramming a fast, complex processor often presents a dilemma, since the technology used for the microprogram memory is often used for the main program memory. But a microprogram needs to run 5 to 10 times as fast as the primary memory if all components are to be fully utilized. To be cost effective, microprogramming depends on the microstore memories being cheaper than conventional combinatorial logic. Some circuits may be cheaper and faster built out of conventional sequential components, while others will be cheaper or faster if microprogrammed. It depends on the number of logic states, inputs, and outputs.

3.3.2. SEMICONDUCTORS FOR PROGRAM MEMORY

We naturally expect that semiconductor memory will ultimately replace core for primary program memories, given their relative rates of price decline (60–100% per year versus 30% per year). The crossover time, determined by a function of the basic costs for different producer–consumer pairs, is complex. It includes consideration of whether there is an adequate supply of production people in the labor-intensive core assembly

process, as well as the reliability and service costs. For IBM, with both core and semiconductor memory technology in-house, the cost crossover occurred in 1971; IBM offered semiconductor memory (actually bipolar) on System/370 (Model 145) and System/7 computers. Within DEC, which has cores and core stacks manufacturing, the crossover point has not yet occurred for large memories. It occurred for small memories (less than 16K) in 1975. In 1969 IBM first delivered the 360 Model 85, which used a mixture of 80-nsec bipolar and 1800-nsec core. This general structure, called a *cache* has been used in the PDP–11/70. The effect on core memories has been to prod their development to achieve lower costs. Recent research has shown that it is possible to hold several states (4 to 16) in a single core. A development of this type, in effect, may mean up to three years additional life in core memory systems.

3.3.3. PROGRAM MEMORY CACHING

A cache memory is a small fast associative memory located between the central processor *Pc* and the primary memory *Mp*. Typically, the cache is implemented in bipolar technology while *Mp* is implemented in MOS or magnetic core technology. The cache contains address/data pairs consisting of an *Mp* address and a copy of the contents of that address. When the *Pc* references *Mp*, the address is compared against the addresses stored in the cache. If there is a match, then *Mp* is not accessed, rather the datum is retrieved directly from the cache. If there is no match, then *Mp* is accessed as usual. Generally, when an address is not found in the cache, it is placed there by the "not found" circuitry, thereby bumping some other address that was in the cache. Since programs frequently cluster their memory references locally, even small caches provide a large improvement over the basic speed of *Mp*.

A significant advantage of a cache is that it is totally transparent to all programs; no software changes need be made to take advantage of it. The PDP–11/70 uses a cache to improve on memory speed.

4. PEOPLE: BUILDERS OF THE DESIGN

Any design project, whether for computers or washing machines, is shaped by the skill, style, and habit of its designers. In this section, we shall outline the evolutionary process of the PDP–11 design, describing how the flavor of the designs was subtly determined by the style of the designers.

A barely minimal computer, i.e., one that has a program counter and a few instructions and that can theoretically be programmed to compute any computable function, can be built trivially. From this minimal point,

performance increases. Eventually the size increases and the design becomes unbuildable or nearly unprogrammable, and therefore not marketable. The designer's job is to find a point on the cost/performance curve representing a reasonable and marketable design, and produce the machine corresponding to that point. There is a nearly universal tendency of designers to $n + 1$ their systems: incrementally improve the design forever. No design is so perfect that a redesign cannot improve it.

The problems faced by computer designers can usually be attributed to one of two causes: inexperience or second-systemitis. *Inexperience* is just a problem of resources: Are there designers available? What are their backgrounds? Can a small group work effectively on architectural specifications? Perhaps most important is the principle that no matter who the architect might be, the design must be clearly understood by at least one person. As long as there is one person who clearly understands the total design, a project can usually succeed with many inexperienced designers. *Second-systemitis* is the tendency of many designers to specify a system that solves all of the problems faced by prior systems—and borders on the unbuildable.

We can see the results of second-systemitis in the history of the PDP–8 implementation designs: alternate designs were bigger, then cheaper. The big designs solved the performance problems of the smaller ones; then the smaller ones solved the cost problems of the bigger ones.

4.1. The System Architecture

Some of the initial work on the architecture of the PDP–11 was done at Carnegie-Mellon University by Harold McFarland and Gordon Bell. Two of the useful ideas, the UNIBUS and the generalized use of the program registers (such as for stack pointers and program counters), came out of earlier work by Gordon Bell and were described in Bell and Newell [71]. The detailed design specification was the work of Harold McFarland and Roger Cady.

The PDP–11/20 was the first model designed. Its design and implementation took place more or less in parallel, but with far less interaction between architect and builder than for previous DEC designs, where the first architect was the implementor. As a result, some of the architectural specifications caused problems in subsequent designs, especially in the area of microprogramming.

As there began to appear other models besides the original Model 20, strong architectural controls disappeared; there was no one person responsible for the family-wide design. A similar loss of control occurred in the design of the peripherals after the basic design.

4.2. A Chronology of the Design

The internal organization of DEC design groups has through the years oscillated between market orientation and product orientation. Since the company has been growing at a rate of 30 to 40% a year, there has been a constant need for reorganization. At any given time, one third of the staff has been with the company less than a year.

At the time of the PDP–11 design, the company was structured along product lines. The design talent in the company was organized into tight groups: the PDP–10 group, the PDP–15 (an 18-bit machine) group, the PDP–8 group, an ad hoc PDP–8/S subgroup, and the LINC–8 group. Each group included marketing and engineering people responsible for designing a product, software and hardware. As a result of this organization, architectural experience was diffused among the groups, and there was little understanding of the notion of a range of products.

The PDP–10 group was the strongest group in the company. They built large, powerful time-shared machines. It was essentially a separate division of the company, with little or no interaction with the other groups. Although the PDP–10 group as a whole had the best understanding of system architectural controls, they had no notion of system range, and were only interested in building higher-performance computers.

The PDP–15 group was relatively strong, and was an obvious choice to build the new mid-range 16-bit PDP–11. The PDP–15 series was a constant-cost series that tended to be optimized for cost performance. However, the PDP–11 represented direct competition with their existing line. Further, the engineering leadership of that group changed from one implementation to the next, and thus there was little notion of architectural continuity or range.

The PDP–8 group was a close-knit group who did not communicate very much with the rest of the company. They had a fair understanding of architecture, and were oriented toward producing minimal-cost designs with an occasional high-performance model. The PDP–8/S "group" was actually one person, someone outside the regular PDP–8 group. The PDP–8/S was an attempt to build a much lower-cost version of the PDP–8 and show the group engineers how it should be done. The 8/S worked, but it was not terribly successful because it sacrificed too much performance in the interests of economy.

The LINC–8 group produced machines aimed at the biomedical and laboratory market, and had the greatest engineering strength outside the PDP–10 group. The LINC–8 people were really the most systems oriented. The LINC design came originally from MIT's Lincoln Laboratory, and there was dissent in the company as to whether DEC should continue to build it or to switch software to the PDP–8.

The first design work for a 16-bit computer was carried out under the eye of the PDP–15 manager, a marketing person with engineering background. This first design was called PDP–X, and included specification for a range of machines. As a range architecture, it was better designed than the later PDP–11, but was not otherwise particularly innovative. Unfortunately, this group managed to convince management that their design was potentially as complex as the PDP–10 (which it was not), and thus ensured its demise, since no one wanted another large computer unrelated to the company's main large computer. In retrospect, the people involved in designing PDP–X were apparently working simultaneously on the design of Data General.

As the PDP–X project folded, the DCM (Desk Calculator Machine, a code name chosen for security) was started. Design and planning were in disarray, as Data General had been formed and was competing with the PDP–8, using a very small 16-bit computer. Work on the DCM progressed for several months, culminating in a design review at Carnegie-Mellon University in late 1969. The DCM review took only a few minutes; the general feeling was that the machine was dull and would be hard to program. Although its benchmark results were good, we now believe that it had been tuned to the benchmarks and would not have fared well on other sorts of problems.

One of the DCM designers, Harold McFarland, brought along the kernel of an alternative design, which ultimately grew into the PDP–11. Several people worked on the design all weekend, and ended by recommending a switch to the new design. The machine soon entered the design-review cycle, each step being an $n + 1$ of the previous one. As part of the design cycle, it was necessary to ensure that the design could achieve a wide cost/performance range. The only safe way to design a range is to simultaneously do both the high- and low-end designs. The 11/40 design was started right after the 11/20, although it was the last to come on the market. The low and high ends had higher priority to get into production, as they extended the market.

Meanwhile an implementation was underway, led by Jim O'Laughlin. The logic design was conventional, and the design was hampered by the holdover of ideas and circuit boards from the DCM. As ideas were tested on the implementation model, various design changes were proposed; for example, the opcodes were adjusted and the UNIBUS width was increased with an extra set of address lines.

With the introduction of large read-only memories, various follow-on designs to the Model 20 were possible. Figure 2 sketches the cost of various models over time, showing lines of constant performance. The graphs show clearly the differing design styles used in the different models.

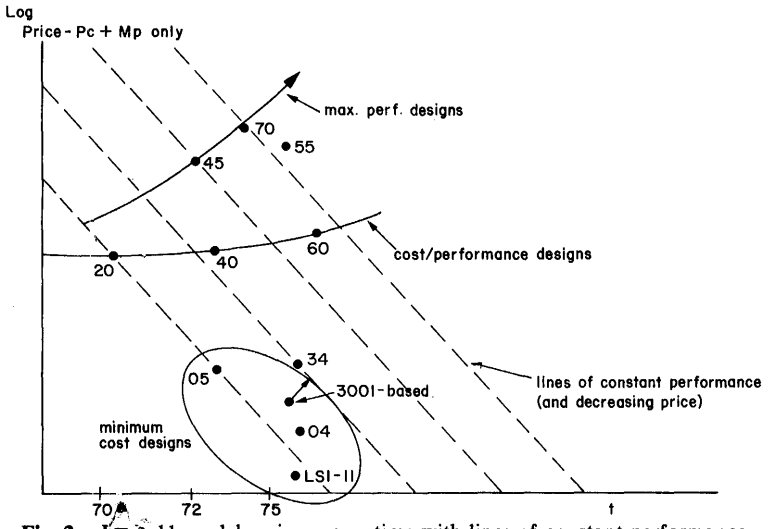


Fig. 2. PDP-11 models price versus time with lines of constant performance.

The 11/40 and 11/45 design groups went through extensive "buy-in" processes, as they each came to the PDP–11 by first proposing alternative designs. The people who ultimately formed the 11/45 group had started by proposing a PDP–11-like 18-bit machine with roots in the PDP–15. Later a totally different design was proposed, with roots in the LINC group, that was instruction subset-compatible at the source program level. As the groups considered the impact of their changes on software needs, they rapidly joined the mainstream of the PDP–11 design.

Note from Fig. 2 that the minimum-cost group had two successors to their original design, one cheaper with slightly improved performance, the other the same price with greatly improved performance and flexibility.

5. THE PDP–11: AN EVALUATION

The end product of the PDP–11 design is the computer itself, and in the evolution of the architecture we can see images of the evolution of ideas. In this section, we outline the architectural evolution, with a special emphasis on the UNIBUS.

In general, the UNIBUS has behaved beyond all expectations. Several hundred types of memories and peripherals have been interfaced to it; it has become a standard architectural component of systems in the $3K to $100K price range (1975). The UNIBUS is a price and performance equalizer: it limits the performance of the fastest machines and penalizes

the lower-performance machines with a higher cost. For larger systems, supplementary buses were added for *Pc–Mp* and *Mp–Ms* traffic. For very small systems like the LSI–11, a narrower bus (called a Q-bus) was designed.

The UNIBUS, as a standard, has provided an architectural component for easily configuring systems. Any company, not just DEC, can easily build components that interface to the bus. Good buses make good engineering neighbors, since people can concentrate on structured design. Indeed, the UNIBUS has created a secondary industry providing alternative sources of supply for memories and peripherals. With the exception of the IBM 360 Multiplexor/Selector bus, the UNIBUS is the most widely used computer interconnection standard.

5.1. The Architecture and the UNIBUS

The UNIBUS is the architectural component that connects together all of the other major components. It is the vehicle over which data flow takes place. Its structure is shown in Fig. 3. Traffic between any pair of components moves along the UNIBUS. The original design anticipated the following traffic flows.

1. *Pc–Mp* for the processor's programs and data.
2. *Pc–K* for the processor to issue I/O commands to the controller *K*.
3. *K–Pc*, for the controller *K* to interrupt the *Pc*.
4. *Pc–K* for direct transmission of data from a controller to *Mp* under control of the *Pc*.
5. *K–Mp* for direct transmission of data from a controller to *Mp*; i.e., DMA data transfer.
6. *K–T–K–Ms*, for direct transmission of data from a device to secondary memory without intervening *Mp* buffering; e.g., a disk refreshing a CRT.

Experience has shown that paths 1 through 5 are used in every system that has a DMA (direct memory access) device. An additional communications path has proved useful: demons, i.e., special *Kio/Pio/Cio* com-

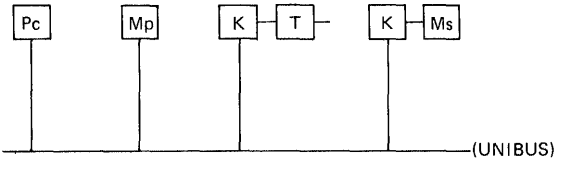


Fig. 3. UNIBUS structure.

municating with a conventional *K*. These demons are used for direct control of another *K* in order to remove the processing load from *Pc*.

Several examples of a demon come to mind: a *K* that handles all communication with a conventional subordinate *Kio* (e.g., an A/D converter interface or communications line); a full processor executing from *Mp* a program to control *K*; or a complete I/O computer, *Cio*, which has a program in its local memory and which uses *Mp* to communicate with *Pc*. Effectively, *Pc* and the demon act together, and the UNIBUS connects them. Demons provide a means of gracefully off-loading the *Pc* by adding components, and is useful for handling the trivial pre-processing found in analog communications, and process-control I/O.

5.1.1. UNEXPECTED BENEFITS FROM THE DESIGN

The UNIBUS has turned out to be invaluable as an "umbilical cord" for factory diagnostic and checkout procedures. Although such a capability was not part of the original design, the UNIBUS is almost capable of dominating the *Pc*, *Tk*'s, and *Mp* during factory checkout and diagnostic work.

Ideally, the scheme would let all registers be accessed during full operation. This is now possible for all devices except *Pc*. By having all *Pc* registers available for reading and writing in the same way that they are now available from the console switches, a second system could fully monitor the computer in the same fashion as a human. Although the DEC factory uses a UNIBUS umbilical cord to watch systems under test, human intervention is occasionally required.

In most recent PDP–11 models, a serial communications line is connected to the console, so that a program may remotely examine or change any information that a human operator could examine or change from the front panel, even when the system is not running.

5.1.2 DIFFICULTIES WITH THE DESIGN

The UNIBUS design is not without problems. Although two of the bus bits were in the original design set aside as parity bits, they have not been widely used as such. Memory parity was implemented directly in the memory; this phenomenon is a good example of the sorts of problems encountered in engineering optimization. The trading of bus parity for memory parity exchanged higher hardware cost and decreased performance for decreased service cost and better data integrity. Since engineers are usually judged on how well they achieve production cost goals, parity transmission is an obvious choice to pare from a design, since it increases

the cost and decreases the performance. As logic costs decrease and pressure to include warranty costs as part of the product design cost increases, the decision to transmit parity might be reconsidered.

Early attempts to build multiprocessor structures (by mapping the address space of one UNIBUS onto the memory of another) were beset with deadlock problems. The UNIBUS design does not allow more than one master at a time. Successful multiprocessors required much more sophisticated sharing mechanisms than this UNIBUS Window.

At the time the UNIBUS was designed, it was felt that allowing 4K bytes of the address space for I/O control registers was more than enough. However, so many different devices have been interfaced to the bus over the years that it is no longer possible to assign unique addresses to every device. The architectural group has thus been saddled with the chore of device address bookkeeping. Many solutions have been proposed, but none was soon enough; as a result, they are all so costly that it is cheaper just to live with the problem and the attendant inconvenience.

5.2. UNIBUS Cost and Performance

Although performance is always a design goal, so is low cost; the two goals conflict directly. The UNIBUS has turned out to be nearly optimum over a wide range of products. However, in the smallest system, we introduced the Q-bus, which uses about half the number of conductors. For the largest systems, we use a separate 32-bit data path between processor and memory, although the UNIBUS is still used for communication with most I/O controllers. The UNIBUS slows down the high-performance machines and increases the cost of low-performance machines; it is optimum over the middle range.

There are several attributes of a bus that affect its cost and performance. One factor affecting performance is simply the data rate of a single conductor. There is a direct tradeoff among cost, performance, and reliability. Shannon [48] gives a relationship between the fundamental signal bandwidth of a link and the error rate (signal-to-noise ratio) and data rate. The performance and cost of a bus are also affected by its length. Longer cables cost proportionately more, and the longer propagation times necessitate more complex circuitry to drive the bus.

Since a single-conductor link has a fixed data rate, the number of conductors affects the net speed of a bus. The cost of a bus is directly proportional to the number of conductors. For a given number of wires, time-domain multiplexing and data encoding can be used to trade perfor-

mance and logical complexity. Since logic technology is advancing faster than wiring technology, we suspect that fewer conductors will be used in all future systems. There is also a point at which time-domain multiplexing impacts performance.

If during the original design of the UNIBUS we could have forseen the wide range of applications to which it would be applied, its design would have been different. Individual controllers might have been reduced in complexity by more central control. For the largest and smallest systems, it would have been useful to have a bus that could be contracted or expanded by multiplexing or expanding the number of conductors.

The cost-effective success of the UNIBUS is due in large part to the high correlation between memory size, number of address bits, I/O traffic, and processor speed. Amdahl's rule of thumb for IBM computers is that 1 byte of memory and 1 byte/sec of I/O are required for each instruction/sec. For DEC applications, with emphasis in the scientific and control applications, there is more computation required per memory word. Further, the PDP-11 instruction sets do not contain the complex instructions typical of IBM computers, so a larger number of instructions will be executed to accomplish the same task. Hence, we assume 1 byte of memory for each 2 instructions/sec, and that 1 byte/sec of I/O occurs for each instruction/sec.

In the PDP-11, an average instruction accesses 3–5 bytes of memory, so assuming 1 byte of I/O for each instruction/sec, there are 4–6 bytes of memory accessed on the average for each instruction/sec. Therefore, a bus that can support 2 megabyte/sec traffic permits instruction execution rates of 0.33–0.5 megainstructions/sec. This implies memory sizes of 0.16–0.25 megabytes; the maximum allowable memory is 0.064–0.256 megabytes. By using a cache memory on the processor, the effective memory processor rate can be increased to balance the system further. If fast floating point instructions were added to the instruction set, the balance would approach that used by IBM and thereby require more memory (seen in the 11/70).

5.3. Evolution of the Design

The market life of a computer is determined in part by how well the design can gracefully evolve to accommodate new technologies, innovations, and market demands. As component prices decrease, the price of the computer can be lowered, and by compatible improvements to the design (the "mid-life kicker"), the useful life can be extended. An example of a mid-life kicker is the writable control store for user microprogramming of

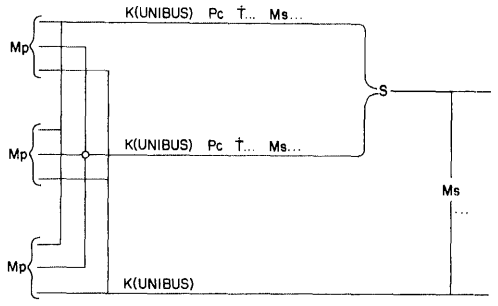


Fig. 4. Use of dual *Pc* multiprocessor system with processorless UNIBUS for I/O data transmission (from Bell *et al.* [70]).

the 11/40 [Almes *et al.* 75]. The PDP–11 designs have used the mid-life kicker technique occasionally. In retrospect, this was probably poor planning. Now that we understand the problem of extending a machine's useful life, this capability can be more easily designed in.

In the original PDP–11 paper [Bell *et al.* 70], it was forecast that there would evolve models with increased performance, and that the means to achieve this increased performance would include wider data paths, multiprocessors, and separate data and control buses for I/O transfers. Nearly all of these devices have been used, though not always in the style that had been expected.

Figure 4 shows a dual-processor system as originally suggested. A number of systems of this type have been built, but without the separate I/O data and control buses, and with minimal sharing of *Mp*. The switch *S* permitting two computers to access a single UNIBUS, has been widely used in high-availability high-performance systems.

In designing higher-performance models, additional buses were added so

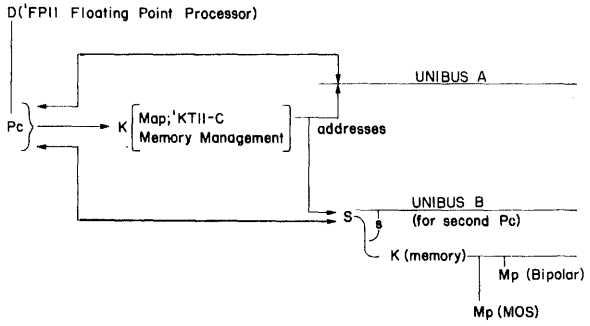


Fig. 5. PMS structure of 11/45.

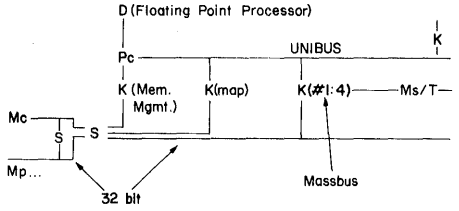


Fig. 6. PMS structure of 11/70.

that a processor could access fast local memory. The original design never anticipated the availability of large fast semiconductor memories. In the past, high-performance machines have parlayed modest gains in component technology into substantially more performance by making architectural changes based on the new component technologies. This was the case with both the PDP–11/45 (see Fig. 5) and the PDP–11/70 (see Fig. 6).

In the PDP–11/45, a separate bus was added for direct access to either

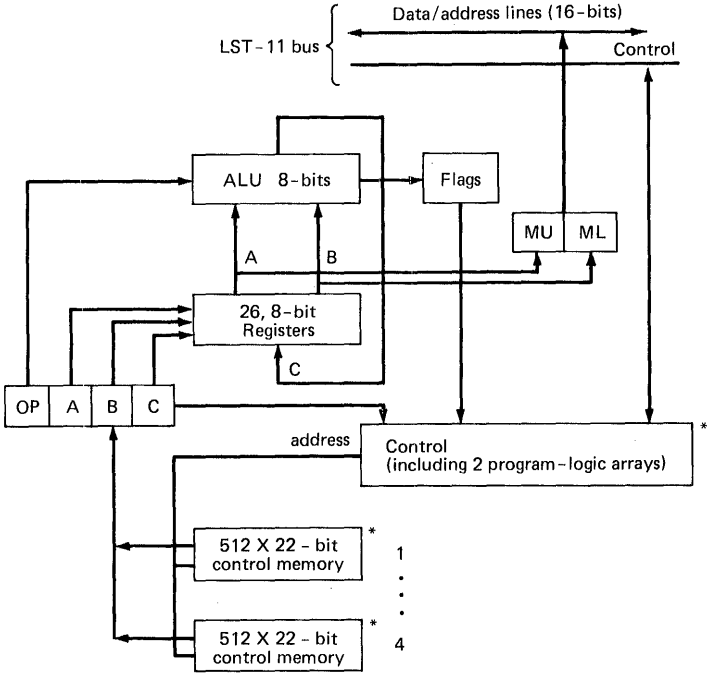


Fig. 7a. PDP – 11/03 (LSI-11) block diagram. (* indicates one LSI chip each and one for data and registers.)

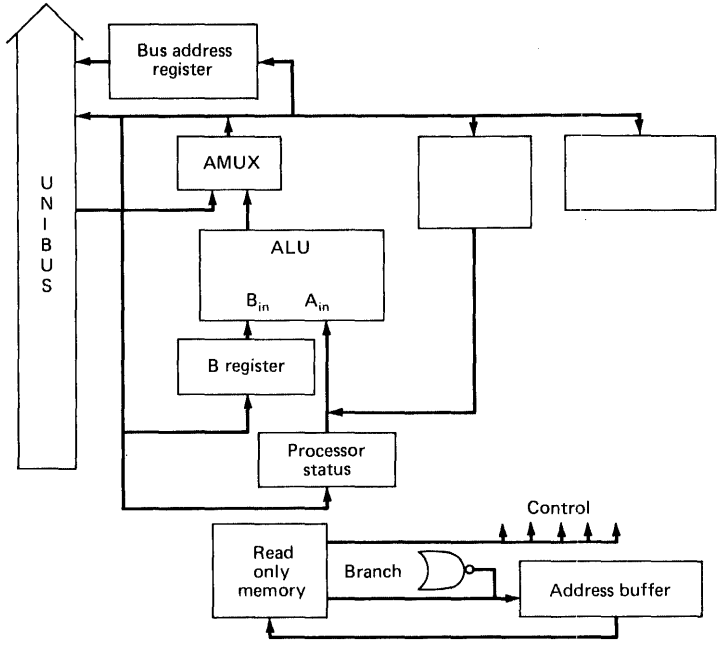


Fig. 7b. PDP-11/05 block diagram.

300-nsec bipolar or 350-nsec MOS memory. It was assumed that these memories would be small, and that the user would move the important parts of his program into the fast memory for direct execution. The 11/45 also provided a second UNIBUS for direct transmission of data to the fast memory without processor interference. The 11/45 also used a second autonomous data operation unit called a Floating Point Processor (not a true processor), which allowed integer and floating-point calculations to proceed concurrently.

The PDP-11/70 derives its speed from the cache, which allows it to take advantage of fast local memories without requiring the program to shuffle data in and out of them. The 11/70 has a memory path width of 32 bits, and has separate buses for the control and data portions of I/O transfer. The performance limitations of the UNIBUS are circumvented; the second *Mp* system permits transfers of up to 5 megabytes/sec., 2.5 times the UNIBUS limit. If direct memory access devices are placed on the UNIBUS, their address space is mapped into a portion of the larger physical address space, thereby allowing a virtual-system user to run real devices.

Figure 7 shows the block diagrams of the LSI-11, the 11/05, and the 11/45. It includes the smallest and largest (except the 11/70) models. Note

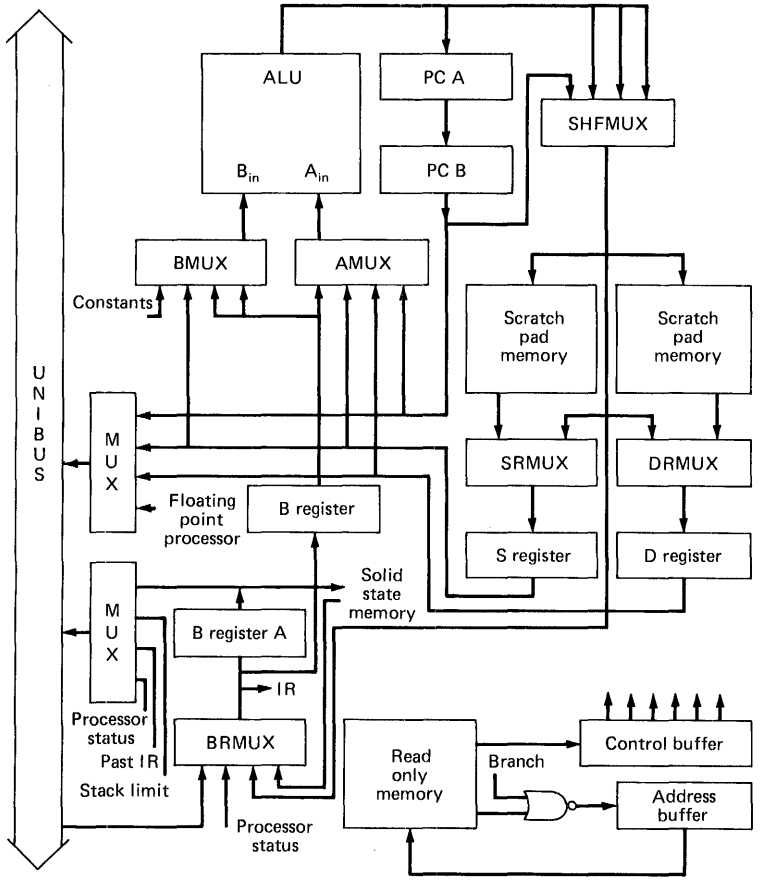


Fig. 7c. PDP-11/45 block diagram.

that the 11/45 block diagram does not include the floating-point operations, but does show the path to fast local memory. It has duplicate sets of registers, even to a separate program counter. The local *Mp.MOS* and *Mp.Bipolar* provide the greatest performance improvements by avoiding the UNIBUS protocols. When only core memory is used, the 11/45 floating-point performance is only twice that of the 11/40. Table III charts the implementation of each design and its performance and parallelism as measured by the microprogram memory width. Note that the brute-force speed of the 11/45 core is only 2 to 4 times faster than the 11/05 for simple data types, i.e., for the basic instruction set. The 11/45 has roughly twice the number of flip-flops.

TABLE III

Model	First delivery	ROM	RAM	Machine time (nsec)	Word length micromem.	No. of microwords	Relative perform. Single prec. fl. pt.	Relative perform. Simple arith.	Mp	Innovations
03 LSI–11	6/75	1k +PLA NMOS; 150 nsec.	13W NMOS; 75 nsec	350	22	1024	3.2	1.5	MOS/core	LSI—4 chips; ODT; maint; fl. pt.
04	9/75	256, 1k; 2k; 50 nsec	4 × 16; 50 nsec	260	38	249	1.6	2.3	MOS/core	Size; maint./test; built-in ASCII console
05	6/72	256; 1k; 50 nsec	4 × 16	137,314 630	40	249	1.4	2.1	Core	Size
20	6/70	—	4 ×16	—	—	—	—	—	Core	ISP; UNIBUS
34	9/75	See 04	4 × 16	200,260	48	470	2.2	3	See 04	Size; modularity
40	1/73	See 05	4 × 16	140,200, 300	56	256	6	2.8	Core	General purpose emulation; fl. pt.
45	6/72	See 05	256 (mapping)	150	64	256	13	3.8	Core	Fastest fl. pt./mem. mgmt. Schottky–TTL
55	6/72 6/76	See 05	256 bipolar 1k bipolar	150	64	256	90	35	Bipolar	Bipolar
70	3/75	See 05	1k cache	150	64	256	85	31	Core	Cache; Systems-oriented
3001	9/75	1k bipolar	11W	170	32	512	—	—	Core	Emulation
C/D	—	—	—	?	?	—	—	—	—	Writable control store

5.4. ISP Design

Designing the ISP level of a machine—that collection of characteristics such as instruction set, addressing modes, trap and interrupt sequences, register organization, and other features visible to a programmer of the bare machine—is an extremely difficult problem. One has to consider the performance (and price) ranges of the machine family as well as the intended applications, and there are always difficult tradeoffs. For example, a wide performance range argues for different encodings over the range. For small systems a byte-oriented approach with small addresses is optimal, whereas larger systems require more operation codes, more registers, and larger addresses. Thus, for larger machines, instruction coding efficiency can be traded for performance.

The PDP–11 was originally conceived as a small machine, but over time its range was gradually extended so that there is now a factor of 500 in price ($500 to $250,000) and memory size (8K bytes to 4 megabytes) between the smallest and largest models. This range compares favorably with the range of the 360 family (4K bytes to 4 megabytes). Needless to say, a number of problems have arisen as the basic design was extended.

For one thing, the initial design did not have enough opcode space to accommodate instructions for new data types. Ideally, the complete set of operation codes should have been specified at initial design time so that extensions would have fit. Using this approach, the uninterpreted operation codes could have been used to call the various operation functions (e.g., floating-point add). This would have avoided the proliferation of runtime support systems for the various hardware/software floating point arithmetic methods (Extended Arithmetic Element, Extended Instruction Set, Floating Instruction Set, Floating Point Processor). This technique was used in the Atlas and SDS designs, but most computer designers don't remember the techniques. By not specifying the ISP at the initial design, completeness and orthogonality have been sacrificed.

At the time the 11/45 was designed, several extension schemes were examined: an escape mode to add the floating point operations, bringing the 11 back to being a more conventional general-register machine by reducing the number of addressing modes, and finally, typing the data by adding a global mode that could be switched to select floating point instead of byte operations for the same opcodes. The FPP of the PDP–11/45 is a version of the second alternative.

It also became necessary to do something about the small address space of the processor. The UNIBUS limits the physical memory to 262,144 bytes (addressable by 18-bits). In the implementation of the 11/70, the physical address was extended to 4 megabytes by providing a UNIBUS

map so that devices in a 256K UNIBUS space could transfer into the 4 megabyte space via mapping registers. While the physical address limits are acceptable for both the UNIBUS and larger systems, the address for a single program is still confined to an instantaneous space of 16 bits, the user virtual address. The main method of dealing with relatively small addresses is via process-oriented operating systems that handle many small tasks. This is a trend in operating systems, especially for process control and transaction processing. It does, however, enforce a structuring discipline in (user) program organization. The RSX series operating systems for the PDP–11 are organized this way, and the need for large addresses is minimized.

The initial memory management proposal to extend the virtual memory was predicted on dynamic, rather than static assignment of memory segment registers. In the current memory management scheme, the address registers are usually considered to be static for a task (although some operating systems provide functions to get additional segments dynamically).

With dynamic assignment, a user can address a number of segment names, via a table, and directly load the appropriate segment registers. The segment registers act to concatenate additional address bits in a base address fashion. There have been other schemes proposed that extend the addresses by extending the length of the general registers—of course, extended addresses propagate throughout the design and include double length address variables. In effect, the extended part is loaded with a base address.

With larger machines and process-oriented operating systems, the context switching time becomes an important performance factor. By providing additional registers for more processes, the time (overhead) to switch context from a process (task) to another process can be reduced. This option has not been used in the implementations of the 11's to date. Various alternatives have been suggested, and to accomplish this most effectively requires additional operators to handle the many aspects of process scheduling. This extension appears to be relatively unimportant since the range of computers coupled with networks tend to alleviate the need by increasing the real parallelism (as opposed to the apparent parallelism) by having various independent processors work on the separate processes in parallel. The extensions of the 11 for better control of I/O devices is clearly more important in terms of improved performance.

The criteria used to decide whether or not to include a particular capability in an instruction set are highly variable and border on the artistic. We ask that the machine appear elegant, where elegance is a combined quality of instruction formats relating to mnemonic significance,

operator/data-type completeness and orthogonality, and addressing consistency. Having completely general facilities (e.g., registers) which are not context dependent assists in minimizing the number of instruction types, and greatly aids in increasing understandability (and usefulness). We feel the 11 provided this.

Techniques for generating code by the human and compiler vary widely and thus affect ISP design. The 11 provides more addressing modes than nearly any other computer. The 8 modes for source and destination with dyadic operators provide what amounts to 64 possible add instructions. By associating the Program Counter and Stack Pointer registers with the modes, even more data accessing methods are provided. For example, 18 varieties of the MOVE instruction can be distinguished [Bell *et al.* 70] as the machine is used in two-address, general-register and stack machine program forms. (There is a price for this generality—namely, fewer bits could have been used to encode the address modes that are actually used most of the time.)

In general, the 11 has been used mostly as a general register machine. In one case, it was observed that a user who previously used a 1-accumulator computer (e.g., PDP–8), continued to do so. Normally, the machine is used as a memory to registers machine. This provides the greatest performance, and the cost (in terms of bits) is the same as when used as a stack machine. Some compilers, particularly the early ones, are stack oriented since the code production is easier. Note, that in principle, and with much care, a fast stack machine could be constructed. However, since most stack machines use Mp for the stack, there is a loss of performance even if the top of the stack is cached. The stack machine is perhaps the most poorly understood concept in computing. While a stack is natural (and necessary) structure to interpret the nested block structure languages, it doesn't necessarily follow that the interpretation of all statements should occur in the context of the stack. In particular, the predominance of register transfer statements are of the simple 2-and 3-address forms

$$D \leftarrow S$$

and

$$D1(index\ 1) \leftarrow f(S2(index\ 2), S3(index\ 3)).$$

These don't require the stack organization. In effect, appropriate assignment allows a general register machine to be used as a stack machine for most cases of expression evaluation. It has the advantage of providing temporary, random access to common sub-expressions, a capability that is usually hard to exploit in stack architectures.

5.5. Multiprocessors

Although it is not surprising that multiprocessors have not been used save in highly specialized applications, it is depressing. One way to extend the range of a family is to build multiprocessors. In this section we examine some factors affecting the design and implementation of multiprocessors, and their affect on the PDP–11.

It is the nature of engineering to be conservative. Given that there are already a number of risks involved in bringing a product to the market, it is not clear why one should build a higher-risk structure that may require a new way of programming. What has resulted is a sort of deadlock situation: we cannot learn how to program multiprocessors until such machines exist, but we won't build the machine until we are sure that there will be a demand for it, i.e., that the programs will be ready.

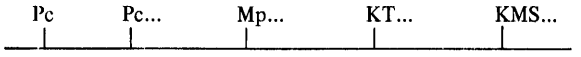
While on the subject of demand for multiprocessors, we should note that there is little or no market pressure for them. Most users don't even know that multiprocessors exist. Even though multiprocessors are used extensively in the high-performance systems built by Burroughs, DEC (PDP–10), and Univac, the concept has not yet been blessed by IBM.

One reason that there is not a lot of demand for multiprocessors is acceptance of the philosophy that we can always build a better single-processor system. Such a processor achieves performance at the considerable expense of cost of spares, training, reliability, and flexibility. Although a multiprocessor architecture provides a measure of reliability, backup, and system tunability unreachable on a conventional system, the biggest, fastest machines are always uniprocessors.

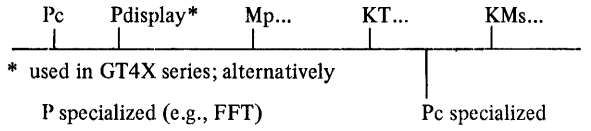
5.5.1. MULTIPROCESSORS BASED ON THE PDP–11

Multiprocessor systems have been built out of PDP–11's. Figure 8 summarizes the design and performance of some of these machines. The topmost structure was built using 11/05 processors, but because of improper arbitration techniques in the processor, the expected performance did not materialize. Table IV shows the expected results for multiple 11/05 processors sharing a single UNIBUS:

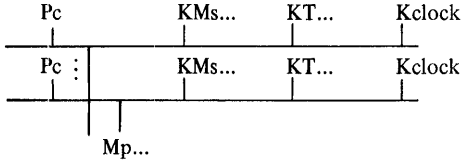
From these results we would expect to use as many as three 11/05 processors to achieve the performance of a Model 40. More than 3 processors will increase the performance at the expense of the cost-effectiveness. This basic structure has been applied on a production basis in the GT4X series of graphics processors. In this scheme, a second *P.display* is added to the UNIBUS for display picture maintenance. A similar structure is used for connecting special signal-processing computers to the UNIBUS, although these structures are technically coupled computers rather than multiprocessors.



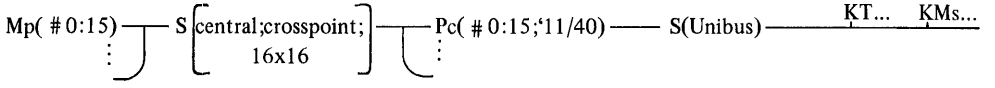
a. Multi-Pc structure using a single Unibus.



b. Pc with P.display using a single Unibus.



c. Multiprocessor using multiport Mp.



d. C.mmp CMU multi-mini-processor computer structure.

Fig. 8. Multiprocessor computer structures implemented using PDP-11.

As an independent check on the validity of this approach, a multiprocessor system has been built, based on the Lockheed SUE [Ornstein *et al.* 72]. This machine, used as a high-speed communications processor, is a hybrid design: it has seven dual-processor computers with each pair sharing a common bus as outlined above. The seven pairs share two shared multiport memories.

TABLE IV

# *Pc*	*Pc* perf. (rel.)	*Pc* price	Price[a]/perf.	SYS price	Price[b]/perf.
1	1.00	1.00	1.00	3.00	1.00
2	1.85	1.23	0.66	3.23	0.58
3	2.4	1.47	0.61	3.47	0.48
40	2.25	1.35	0.60	3.35	0.49

[a] *Pc* cost only.
[b] Total-system cost, assuming one-third of system is *Pc* cost.

The second type of structure given in Fig. 8 is a conventional multiprocessor using multiple-port memories. A number of these systems have been installed, and they operate quite effectively. However, they have only been used for specialized applications.

The most ambitious multiprocessor structure made from PDP–11's, C.mmp, is amply described in the literature [Wulf *et al.* 72]. As it becomes a user machine, we will gather data about its effectiveness. Hopefully, data from this and other multiprocessor efforts will establish multiprocessors as applicable and useful in a wide variety of situations.

6. FUTURE PLANS AND DIRECTIONS

The problems encountered on the PDP–11 project are not peculiar to that machine, or to any machine or style of architecture. In the course of the project, we have isolated several specific problems in computer design. We intend to explore each of them further.

6.1. The Bus Specification Problem

It has taken a long time to understand the UNIBUS in terms of its electrical, performance, and logical capabilities. The existing bus specifications, however inadequate, are the result of many iterations of respecification based on experience and redesign. Several description techniques have been tried: timing diagrams, threaded diagrams showing the cause and effect of signals, and partial state flowcharts showing state in master and slave components. A rigorous specification language, such as BNF, would be helpful. BNF has proven helpful in the specification of communication links, but is too clumsy for general use, and is not widely understood by engineers and programmers.

The most important use of a rigorous bus specification is the testing of faulty components rather than the exercising of good ones. A bus specification would provide a behavior standard against which to check faulty components. It is not clear how one should best attack the problem of bus behavior specification. A safe place to start would be an exhaustive set of examples.

6.2. Characterizing Computation Problems

When a user comes to us with a task needing computerization, we don't have a good way to describe the computational needs of the task. The needs are multidimensional, consisting of the procedural algorithms, the file structure, the interface transducers, reliability, cost, and development

deadline. This communications difficulty exists between computer designers and operating-system designers as much as between computer designers and end users.

Even when there is a good way to specify to the system designer exactly what the user's computational needs might be, there is still a lot of work in finding an architecture to best solve that problem and finding an implementation to best build that architecture.

6.3. Operating Systems

A taxonomy and notation is needed to describe the functions of a system, especially the operating systems. There is no good methodology for talking about tradeoffs, because the functions and structures of a system are so vague.

There exist numerous operating systems for the PDP–11. One of the reasons for this situation is that there is no easy way to compare an existing system with a design for a new one. Instead, an engineering-marketing conspiracy invents a new system because it is oriented toward a particular market in some nebulous way. If we had the ability to specify operating system behavior in a uniform and comprehensible way, then a system could be analyzed before it is programmed.

6.4. Problems with Architectural Range

In a growing family of computers, the designer is constantly faced with the question of whether or not to build a certain model or provide an certain point on the price/performance curve. The decision is colored by technology, user requirements, competitor offerings, and available design staff. It is difficult to answer precisely even a question so simple as whether to build two models that are close together (as the 11/40 and 11/45), or to make a single model and expand it with a multiprocessor option.

The range problem occurs at other levels. Consider memory. The number of memory technologies available is growing constantly, and the once-clear boundaries between memory classes based on memory speed are blurring. Some of the new electronic-based technologies such as CCD and magnetic bubbles have an access time in the 100-microsecond range, and fill the gap between traditional random-access memories (.1 to 1 microsecond) and electromechanical memories like disks or drums (1 millisecond to 100 millisecond). The system designer must decide how much of which kinds of memory will be used in each implementation. It may well be that a solution to problems of this sort will be dependent on the ability to characterize the computational needs.

7. SUMMARY

In this paper we have reexamined the PDP–11 in the light of six years of experience, and have compared its successes and failures with the goals and problems of the initial ideas. With the clarity of hindsight, we now see the initial design problems. Many mistakes were made through ignorance, and many more because the design work was started too late. As we continue to evolve and improve the PDP–11 computer over the next five years, it will indeed be interesting to observe whether the PDP–11 can continue to be a significant, cost-effective minicomputer. We believe it can. The ultimate test is its use.

ACKNOWLEDGMENT

I would like to thank Brian Reid for editing and rewriting sections of this paper.

REFERENCES

Almes, G. T., Drongowski, P. J., and Fuller, S. H., Emulating the Nova on the PDP–11/40: a case study. *Proc. COMPCON, Washington, D.C., September 1975.*

Bell, C. G., Cady, R., McFarland, H., Delagi, B., O'Loughlin, J., Noonan, R., and Wulf, W., A new architecture of minicomputers— The DEC PDP–11. *Proc. SJCC* **36**, 657–675 (1970).

Bell, C. G., and Newell, A., *Computer Structures*. McGraw–Hill, New York, 1971.

Eckhouse, R. H., *Minicomputer Systems: Organization and Programming (PDP–11)*. Prentice–Hall, Englewood Cliffs, New Jersey, 1975.

Gear, C. W., *Computer Organization and Programming, Second Edition*. McGraw–Hill, New York, 1974.

McWilliams, T., Sherwood, W., and Fuller, S., PDP–11 implementation using the Intel 3000 microprocessor chips. *Proc. NCC* **46**, 243–253 (1977).

O'Loughlin, J. F., Microprogramming a fixed architecture machine. *Microprogramming and Systems Architecture Infotech State of the Art Rep.* **23**, 205–224 (1975).

Ornstein, S. M., Heart, F. E., Crowther, W. R., Rising, H. K., Russell, S. B., and Michael, A., The terminal IMP for the ARPA computer network. *Proc. SJCC* **40**, 243–254 (1972).

Shannon, C. E., A mathematical theory of communication. *Bell Sys. Tech. J.* **27**, 379–423, 623–656 (1948).

Stone, H. S., and Siewiorek, D. P., *Introduction to Computer Organization and Data Structures: PDP–11 Edition*. McGraw–Hill, New York, 1975.

Wulf, W. A., and Bell, C. G., C.mmp: a multi-mini-processor. *Proc. FJCC* **41**, 765–778 (1972).