# DEC PDP-10

Computer Systems

G. Bell, S. H. Fuller, and D. Siewiorek, Editors

# The Evolution of the DECsystem 10

C. G. Bell,
Digital Equipment Corporation, Maynard, Mass. and Carnegie-Mellon University, Pittsburgh, Pa.

A. Kotok, T. N. Hastings, and R. Hill
Digital Equipment Corporation, Maynard, Mass.

**The DECsystem 10, also known as the PDP-10, evolved from the PDP-6 (circa 1963) over five generations of implementations to presently include systems covering a price range of five to one. The origin and evolution of the hardware, operating system, and languages are described in terms of technological change, user requirements, and user developments. The PDP-10's contributions to computing technology include: accelerating the transition from batch oriented to time sharing computing systems; transferring hardware technology within DEC (and elsewhere) to minicomputer design and manufacturing; supporting minicomputer hardware and software development; and serving as a model for single user and timeshared interactive minicomputer/microcomputer systems.**

**Key Words and Phrases: computer structures, architecture, operating system, timesharing**

**CR Categories: 4.32, 6.21, 6.3**

## 1. Introduction

The project originating the PDP-6, DECsystem 10, and DECsystem 20 series of scientific, timeshared computers began in the spring of 1963, and continued with the delivery of a PDP-6 in the summer of 1964. Initially, the PDP-6 was designed to extend DEC's line of 18-bit computers by providing more performance at increased price. Although the PDP-6 was not constrained to be a member in a family of compatible computers, the series evolved into five basic designs (PDP-6, KA-10, KI-10, KL10, and KL20) with over 700 systems installed. The notions and need for compatibility were neither understood at the initial design time nor did we have adequate technology to undertake such a task. Each successive implementation in the series has generally offered increased performance for only slightly increased cost. Currently, the KL10 and KL20 systems span a 5 to 1 price range.

TOPS-10, the major user software interface, developed from a 6 Kword monitor for the PDP-6. A second user interface, TOPS-20, introduced in 1976 with upgraded facilities is based on multiprocess operating systems advances.

The paper is divided into seven sections. Section 2 provides a brief historical setting followed by a discussion of the initial project's goals, constraints, and basic design decisions. The instruction set and system organization are given in Sections 4 and 5 respectively. Section 6 discusses the operating system while Section 7 presents the technological influences on the designs. Sections 4 to 7 begin with a presentation of the goals and constraints, proceed to the basic PDP-6 design, and conclude with the evolution (and current state). We try to answer the often asked questions "Why did you do . . .?", by giving the contextual environment. Figure 1 helps summarize this context in the form of a time line that depicts the various hardware/software technologies (above line) and when they were applied (below line) to the DECsystem 10.

## 2. Historical Setting

The PDP-6 was designed for both a timeshared computational environment and real-time laboratory use with straightforward interfacing capability. At the initiation of the project, three timeshared computers were operational: A PDP-1 at Bolt, Beranek and Newman (BBN) which used a high-speed drum that could swap a 4 Kword core image in one 34-millisecond revolution; an IBM 7090 system at MIT, called CTSS, which provided each of 32 users a 32 Kword environment; and an AN/FSQ-32V at SDC which could serve 40 simultaneous users.

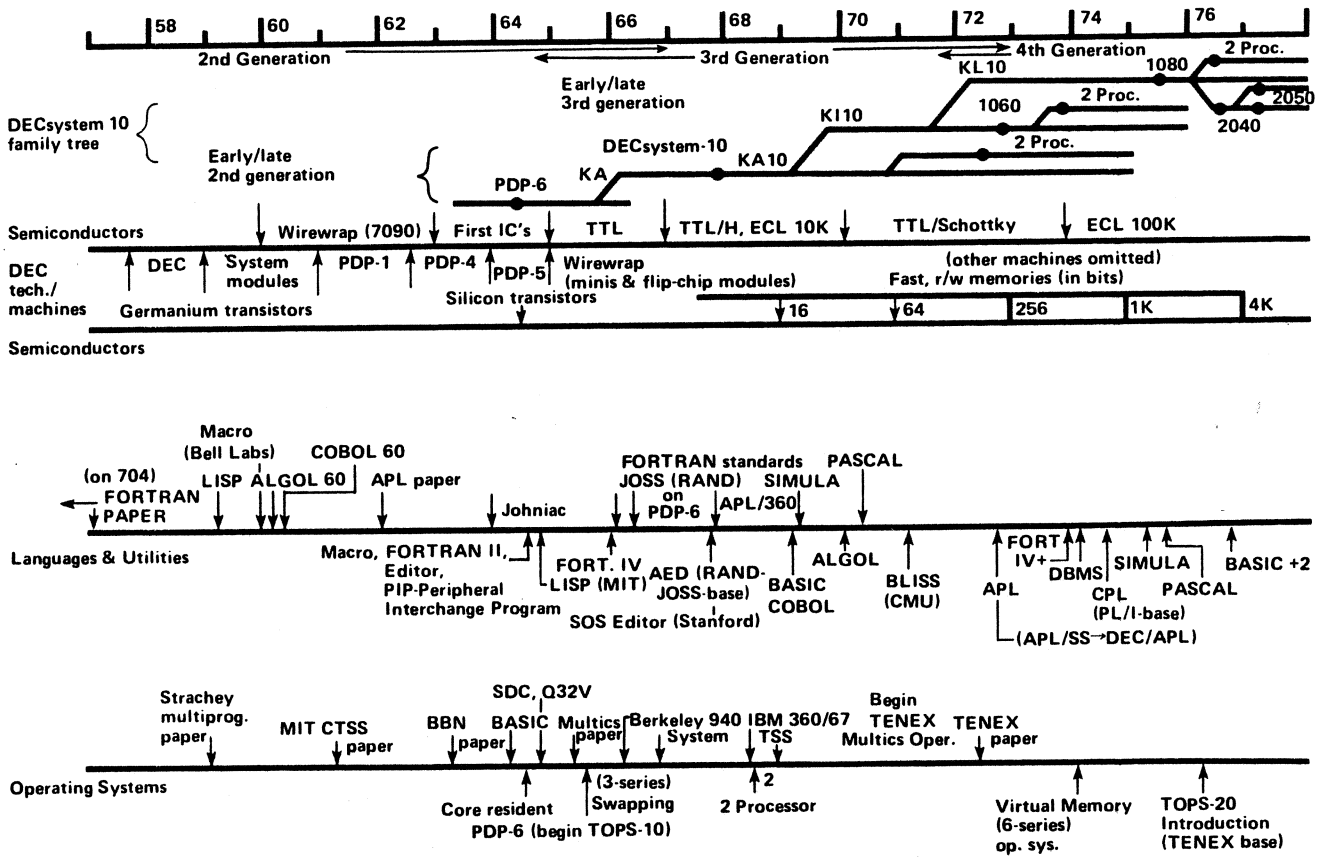
The Bell Laboratory's IBM 7094 Operating System was a model operating system for batch users. Burroughs had implemented a multiprogrammed system on the B5000. Dartmouth was considering the design of a single language, timesharing system which subsequently became Basic. The MIT Multics System, the Berkeley SDS 940, the Stanford PDP-1 based timeshared system for computer aided instruction, and the BBN Tenex System all contributed concepts to the DECsystem 10 evolution in the 1960's.

In architecture, the Manchester Atlas [3, ch. 23] was exemplary, not because it was a large machine



Author's address: Digital Equipment Corporation, 146 Main Street, Maynard, MA 01754.

Fig. 1. Time line diagram of DECsystem 10 evolution.




that we would build, but because it illustrated a number of good design principles. Atlas was multiprogrammed with a well-defined interface between the user and operating system, had a very large address space, and introduced the notion of extra codes to extend the functionality of its instruction-set. Paging was a concept we just could not afford to implement without a fast, small memory. The IBM Channel concept was in use, on their 7094, and was one we wanted to avoid since our minicomputers (e.g. PDP-1) were generally smaller than a single channel and could outperform the 7094 in terms of i/o concurrency and i/o programmability by a clean, simple interrupt mechanism.

The DEC product line in 1964 is summarized in Table I. Corporation wide sales totaled $11 million at that time and it was felt that computers had to be offered in the $20,000 to $300,000 range. We were sensitive to the problems encountered by not having enough address bits, having watched DEC and IBM machines exceed their addressing capacities.

On the software side, most programmers at DEC had been large machine (16 to 32 Kwords) users although they had most recently programmed minicomputers where program size of 4 to 8 Kwords was the main constraint. There was not a good understanding of operating systems structure and design in either academia or industry. For example, MIT's Multics Project was being formed and IBM's 360/TSS Project

Table I. DEC's 1964 computer products.

| Name | Year Introduced | Word Size (Bits) | Price ($K) | Status |
|---|---|---|---|---|
| PDP-1 | 1960 | 18 | 120 | Marketed |
| PDP-2 | 1960 | 24 | – | Reserved for future implementation |
| PDP-3 | 1961 | 36 | – | Paper machine |
| PDP-4 | 1962 | 18 | 60 | Marketed |
| PDP-5 | 1964 | 12 | 27 | Introduced |
| PDP-6 | 1964 | 36 | 300 | Introduced |

didn't start until 1965. Generally there were no people who directly represented the users within the company, although all the designers were computer users. A number of users in the Cambridge (Mass.) community advised on the design (especially John McCarthy, Marvin Minsky, and Peter Sampson at the MIT Artificial Intelligence Laboratory).

Although there was little consensus that Fortran would be so important, it was clear our machine would be used extensively to execute Fortran. The macroassemblers, basically unchanged even today, were used in various laboratories and our first one for PDP-1 was done by MIT in 1961. We also felt the list languages, especially LISP for symbolic processing were important. There was virtually no interest in business data processing although we had all looked at Cobol.

We did not understand the concept of technology

evolution very well, even though integrated circuits were both forecast and under development. Germanium transistors were available, and silicon transistors were just on the market. IBM was using machine wirewrap technology, while DEC back panels were handwired and soldered. The basic DEC logic circuits were saturating transistor as distinct from the more expensive current mode used by IBM in the 7094 and Stretch computers. Production core memories of 2 microseconds were beginning to appear, and their speed was improving. Our PDP-1 used a 5 microsecond core. Hence, it was unclear what memory speed a processor should support.

The notions of compatibility and family range were not appreciated even though SDS (eventually XDS and now nonexistent) had built a range of 24-bit computers. We adhered to the then imposed convention of the word length being a multiple of six bits (the number of bits in the standard character code), but designed the machine to handle arbitrary length characters.

## 3. Overall Goals, Constraints, and Basic Design Decisions

Table II lists the initial goals, constraints, and some basic design decisions. Presentation of this list separately from the design is difficult because the goals and constraints were not formally recorded as such and have to be extracted from design descriptions and our unreliable, and self justifying memories. Table II will be used in discussing the design.

The initial design theme was to provide a powerful, timeshared machine oriented to scientific use, although it subsequently evolved to commercial use. John McCarthy's definition [9] of timesharing, which we subscribed to, included providing each user with the illusion of having his own large computer. Thus our base design provided protection between the users and a mechanism for the common resources to be allocated and controlled. The machine had to also support a variety of compiled and interpreted languages. The construction was to be modular so that it could evolve and users could build large systems including multiprocessors. It should add to the top of DEC's existing line of 12- and 18-bit computers. It should be simple, buildable, and supportable by a small organization. Thus, it should use as much DEC hardware technology as possible.

## 4. The Instruction-Set Processor

Our goals for an ISP were: To efficiently encode the various programs using both compiled and interpreted languages; to be understandable and able to be remembered by its users; to be buildable in current technology at a competitive price; and to permit a compiler to provide efficient program production.

### Data Types and Operators

Earlier DEC designs and the then current 6-bit character standard forced a word length which was a multiple of 6, 12, and 18 bits. Thus a 36-bit word was selected.

The language goals and constraints forced the inclusion of integer and real (floating point) variables. We chose two's complement integer representation rather than the sign-magnitude representation used on the 7090, or the one's complement representation on PDP-1. The floating point format was chosen to be the same as the 7090, but with a format that permitted comparison to be made on the number as an integer in order to speed up comparisons and only require a single set of compare instructions. Special (common) case operators (e.g., $V = 0$, $V = V + 1$, $V = V - 1$) were included to support compiled code. Our desire to execute LISP directly resulted in good address arithmetic. As a result, both LISP and Fortran on DECsystem 10 are encoded efficiently.

Since the computer spends a significant portion of its time executing the operating system, the efficient support of operating system data types is essential. A number of instructions should be provided for manipulating and testing the following data types: Boolean variables (bits); Boolean vectors; arbitrary length field access (load/store only); addresses; programs (loops, branching and subprograms); ordinary integers; and the control of i/o. A significant number of control instructions were included to test addresses and other data types. These tests either controlled flow by a jump or skip of the next instruction (which is usually a jump). Loop control was a most important design consideration.

Table III gives the data types and instructions present in the various implementations. The KA10 and PDP-6 processor instruction sets were essentially the same, but differed in the implementation. The PDP-6 had 365 instructions. A double precision negate instruction in the KA10 improved the subroutine performance for double precision reals. The instruction, *find first one in a bit vector,* was also added to assist operating system resource allocation and to help in a specific application sale (that fell through). Finally, double precision real arithmetic instructions were added to the KI10 using the original PDP-6 programmed scheme. A few minor incompatibilities were introduced in the KI to improve performance.

With the decision to offer Cobol in 1970, better character and decimal string processing support was required from the instruction set. The initial Cobol performance was poor for character and decimal arithmetic because each operation required software character by character conversion to an integer, the operation (in binary or double precision binary), and software reconversion to a character or a decimal number. The KL10 provided much higher performance for Cobol by having the basic instructions for comparing

Table II. Initial goals, constraints, and basic design decisions.

User/Language/Operating System
- Cheaper cost/user via timesharing without inconvenience of batch processing
- Timeshared use via terminals with protection between users
- Independent user machines to execute from any location in physical memory
- Unrestricted use of devices e.g. full duplex use of terminals
- Support for wide range of compiled and interpreted languages
- No special batch mode, batch must appear like terminal via a command file
- Device independent i/o so that programs would run on different configurations and i/o could be shared among the user community
- Direct i/o for real time users
- Primitive command language to avoid need for large internal state
- Minimum usable system <16 Kwords
- Modular software to correspond to modular hardware configurations

Instruction-Set Processor (ISP)
- Support user languages by data types and special operations
  - scientific (i.e., Fortran) ⇒ integers, reals, Boolean
  - list processing (i.e., LISP) ⇒ addresses, characters
  - support recursive and reentrant programming ⇒ stack mechanism
- Support operating systems
  - effective as machine language ⇒ Booleans, addresses, characters, i/o
  - operating system is an extension of hardware via defined operating codes
- Word length would be 36-bits (compatible with DEC's computers)
- Large (1/4 million 36-bit words = 1 million 9-bit bytes) address
- Require minimal hardware ⇒ simple
- General-register based (design decision) with completely general use
- Easy to use and remember machine language
  - orthogonality of addressing (accessing) and operators
  - completeness of operators
  - direct (not base + displacement) addressing
  - few exceptional instructions
- 2's complement arithmetic (multiple precision arithmetic)

PMS Structure
- Maximum modularity so that users could easily configure any system
- Easy to interface
- Asynchronous operation—system must handle evolving technology
- Multiprocessors for incremental and increased performance (2–4 in design)
- No Pio's (IBM Channels), use simple programmed i/o with interrupts and direct memory access for high speed data transmission

Implementation
- Simple; reliable
- Asynchronous logic and busses for speed in light of uncertain logic and memory speed
- All state accessible to field service personnel via lights
- Use DEC (10 mHz vs 5 mHz) circuit/logic technology (manpower constraint)
- Buildable without microprogramming (no fast, read-only, memories in 1963)

Organizational/Marketplace
- Add to high end of DEC's computers
- Use minimal resources, while supporting DEC's minicomputer efforts

character and decimal strings—where a character can be a variable size. For arithmetic operations, instructions were added to convert between string and double precision binary. The actual operations are still carried out in binary. For add and subtract, the time is slightly longer than a pure string based instruction, but for multiplying and dividing, the conversion approach is faster.

### Stack vs General Registers Organization

A stack machine was considered based on the B5000 and George Interpreter (which later became the English Electric KDF9). A stack with index register machine was proposed, but rejected on the basis of high cost and fear of poor performance, for executing the operating system, LISP, and Fortran. The compromise we made was to provide a number of instructions to operate on a stack, yet use the general registers as stack pointers.

An interesting outcome of our experience was that one of us (Bell) discovered a more general structure whereby either a stack or general register machine could be implemented by extending addressing modes and using the general registers for stack pointers. This scheme was the basis of the PDP-11 ISP [1].

We currently believe that stack and general register structures are quite similar and tend to be a tradeoff between control (either in a program or in the interpretation of the ISP) and performance. Compilers for general register machines often allocate registers as though they are a stack. Table IV compares the stack and general register approaches.

A general register architecture was selected with the registers in the memory address space. The general registers (multiple accumulators) should permit a wide (general) range of use. Both eight and sixteen were considered. By the time the uses were ennumerated, especially to store inner loops, we believed sixteen

Table III. Data-types of DECsystem 10/20.

| Data type | Length (bits) | Machine | Operators and [#instructions] | Operator Location |
|---|---|---|---|---|
| Boolean | 1 | all | 0,1,−, test by skip [64] | AC ←f(AC) |
| Boolean-vector | 36 | all | all 16 [64] | AC and/or mem ←f(AC,mem) |
| characters | 0-36 = v | all | load, store [5] | AC ↔(mem) |
| character-string | v × n | KL | compare [8]; move [4] | f(mem)=g(mem); mem f(mem) |
| digit-string | v × n | KL | convert to double integer | f(AC) ↔f(men) |
| half word, 2's comp. integers = addresses | 18 | all | load, store [64]; index loop control | AC ↔fmem; AC ←f(AC) |
| full word, 2's comp. integers (and fractions) | 36 | all | load, store, abs., −(negate)[16] +,−,×,/,+1,−1,×rotate test (by skip & jumps) | AC and/or mem←f(AC,mem) |
| double word, 2's comp. integers (and fractions) | 72 | KL | load, store, −(negate)[4]; +,−, ×,/[4] | AC ↔f(mem); AC ←f(AC,mem) |
| real | 9 (exponent) +27(mantissa) | all | load, store, abs., −(negate), +,−,×,/,×[35]; test (by skip, jump) [16] | AC and/or mem ←f(AC,mem) immediate mode was added in KA |
| double real | 9 + 54<br>9 + 63 | KI,KL<br>KI,KL | load, store, abs, negate, +,−,×,/ [8] | KA provided negate instruction |
| word stack | 36 | all | load, store, call, return[4] | Stack ↔ Memory |
| word vector | 36 × k | all | move [1] | mem[a:a+k]←mem[b:b+k] |
| i/o program | 36 | all | short call/return; UUO | AC, mem |

were needed. They could be used as: base and index, set of Booleans (flags), ordinary accumulator and multiplier-quotient (from 7090), subroutine linkage, fast access for temporary and common subexpressions, top of stack when accessed explicitly, pointer to control stacks, and fast registers to hold small programs.

Since the AC's were in the address space, ordinary memory could be used in lieu of fast registers to reduce the minimal machine price. In reality, nearly all users bought fast registers. Eight registers may have been enough. A smaller number would have provided more rapid context switching and assisted the assembly language programmer who tried to optimize (and keep track of) their use. In fact, Lunde [7] has shown that eight working registers would be fine to support the higher level language usage. Multiple register sets were introduced in the KI10 to reduce context-switching time.

Table IV. Comparison of stack and general register architectures.

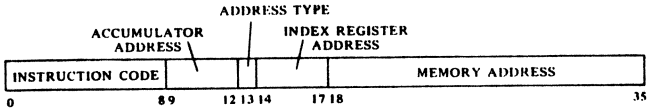
| | Stack | General Register |
|---|---|---|
| Number of registers | approximately the same | |
| Register use | fixed to stack operation | can be arbitrary |
| Control | built in hardward (implicit) | simple, explicit in program when used as a stack |
| Access to local variables | 1 or 2 elements at top of stack | full set in general registers |
| Compiler | easy (no choice) | an assignment (use) problem |
| Program encoding | fewer bits | more bits give access to registers for intermediate and index values |
| Performance | high if element on stack top | high if in general registers (performs relatively better than stack) |

## Instruction-Set Encoding and Layout

The ease of implementation goal forced an instruction-set design style that later turned out to be easy to fabricate with the KL10 microprogram implementation. This also simplified the fabrication of compilers. In fact, of the 222 instructions useful for Fortran data-types, the earliest compiler used 180 of them and the current compiler uses 212. We used three principles, we now understand, for the ISP design:

a. Orthogonality—an address (with index and indirect control fields) is always computed in the same way, independent of the data-type it references. Indirect addressing occurs as long as the instruction addressed has an indirect bit on an indefinite basis.
b. Completeness and symmetry—where possible each arithmetic data type should have a complete and identical set of operations.
c. Mapping among data types—instructions should exist to convert among all data types. Several data types were incomplete (characters, half-words) and these should be converted to data types with a complete operator set.

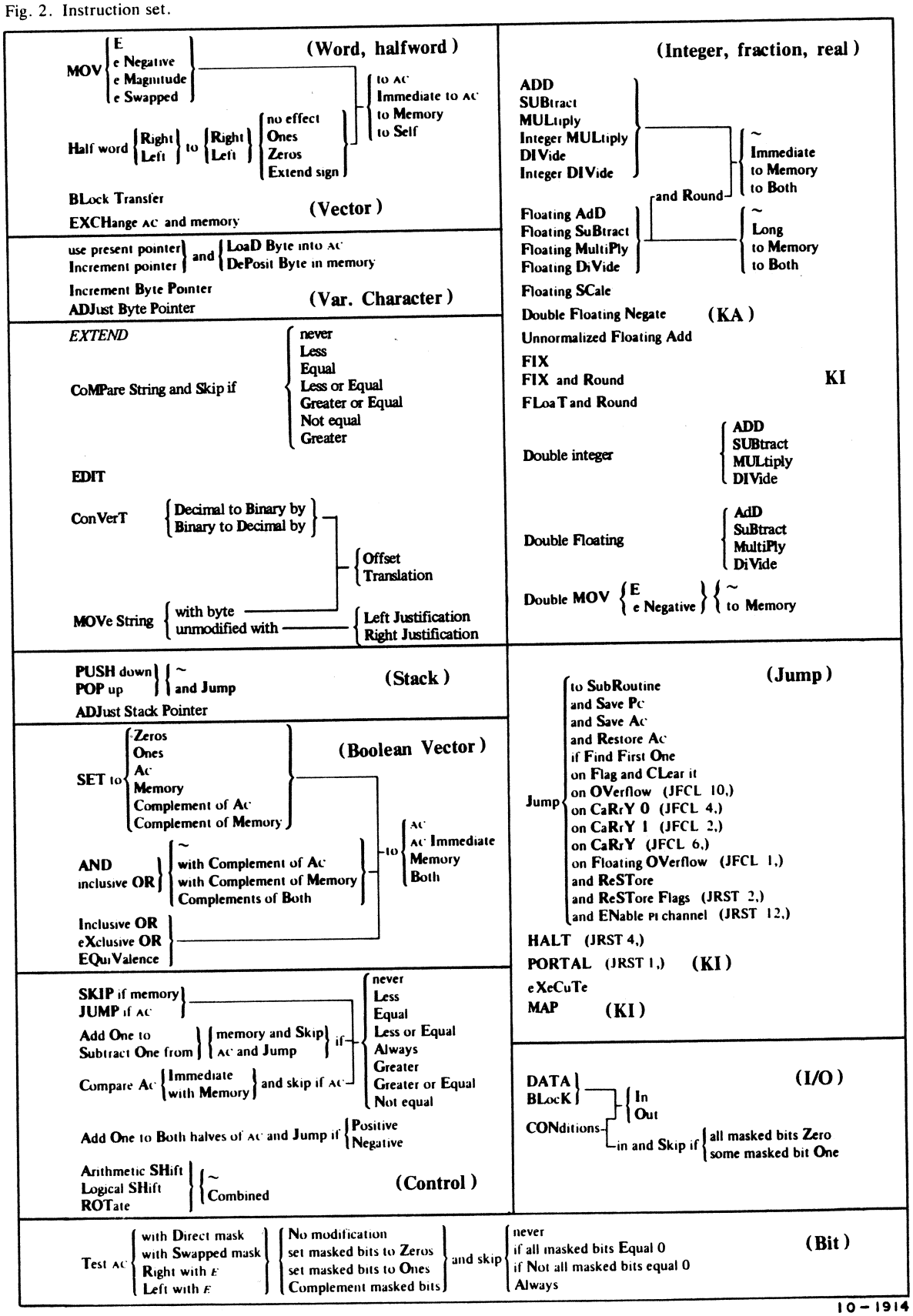
The instruction is mapped into the 36 bit word as follows:




BASIC INSTRUCTION FORMAT

ACCUMULATOR ADDRESS is 1 of 16 Accumulators (General Registers)
INDEX REGISTER ADDRESS is index designator to 1 of 15 AC's
L is indirect bit
MEMORY ADDRESS is address or literal

Fig. 2. Instruction set.




The entire instruction-set fits easily within a single figure (see Figure 2). The boldface letters denote instruction mnemonics. The data types and operations are generally deducible by the instruction names: operator names (e.g. ADD) for word (or integer); D-double integers; H-half world; BL-vector; 16-operator

names (e.g. AND) for Boolean vectors, Test-Boolean (bits); J-jump/skip for program control; F-floating; DF-double floating. The i/o and interrupt instructions are described in the PMS section.

**Multiprogramming/Monitor Facilities**

The initial constraint (circa 1963) of a timeshared computer with a common operating system led to several hardware facilities:

1. two basic machine modes: user and executive (each with different privileges);
2. protection against operations to halt the computer or affect the common i/o when in user mode;
3. communication between the user and operating system for calling i/o and other shared functions; and
4. memory mapping—separation of user programs into different parts of physical memory with protection among the parts and program relocation beyond the control of user.

An executive/user mode was necessary for protection facilities in a shared operating system while providing each user with his own environment. Although there was a temptation (due to having a single operating system) to eliminate or make optional the executive mode and the general registers be, we persevered in the design and now believe this to be an essential part of virtually every computer! (The only other necessary ingredient in every computer is adequate error detection, such as parity.) Separation into at least two separate operating regions (user and executive) also permits the more difficult, time constrained i/o programs to be written once and to have a more formal interface between system utilities and user.

The UUO (Unimplemented User Operation) is an instruction like the Atlas Extracode and IBM 360 SVC to call operating system functions and common user-defined functions. It also calls functions not present in earlier machines. Thus a single operating system could be used (by selecting the appropriate options) over several models. This use appears to be more extensive than in the IBM System 360/370.

The goals of low cost hardware and minimal performance degradation constrained the protection facilities to a single pair of registers to relocate programs in increments of 1 Kwords. Two 8-bit registers (base and limit registers) with two 8-bit adders were required for this solution. Thus each user area was protected while running and a program could be moved within primary or secondary memory (and saved) because user programs were written beginning at location 0. This is identical to the CDC 6600-7600 protection/relocation scheme.

In the KA10 a second pair of registers was added so that the common read-only segment of a user's space could be shared. For example, this enabled one copy of an editor, compiler, or runtime system to be shared among multiple users. Programs were divided into a 128 Kword read-write segment and a 128 Kword read only segment. Since each user's shared segment had to occupy contiguous memory, holes would develop as users with different shared segment requirements were swapped. This led to "core shuffling" and in a busy system up to 2% of the time might be spent in this activity. The operating system was modified in the early 70's at the Stanford Artificial Intelligence Laboratory so that the high, read-only segment could share common, global data. In this way a number of separate user programs could communicate, to effectively extend the program size beyond the 256 Kword limit. In retrospect, instructions to move data more easily between a particular user region and the operating system would have been useful; this was corrected in KI10 and is described below.

With the availability of medium scale integrated circuits, small (32 word) associative memories could be built. This enabled the introduction of a paging scheme in the KI10. Each 512 word page could be declared sharable or private with read-only or read-write access. The basic two mode protection facility was expanded to four modes: Supervisor, Kernel, Public, and Concealed. There were two monitor modes: Kernel mode provides protection for i/o and system functions common to all users; and Supervisor mode is specialized for a single user. The two user modes are: Concealed for proprietary programs, and Public for shared programs. For protection purposes, the modes are only changed at selected entry portals. The page table was more elaborate than that of the Atlas (circa 1960) whose main goal was to provide a one level store whereby large programs could run on small physical memories. In fact the first use of KI10 paging required all programs to be resident rather than having pages being demand driven. A gain over the KA10 was realized by not requiring programs to be in a single contiguous address space. The KI10 design provided more sharing, and increased efficiency over the KA10. The KL10 extended KI10 paging for use in the TOPS 20 operating system to be described later.

## 5. PMS[1] Structure

Table II gives the major goals and constraints in the PMS structure design. This section describes system configurations, the i/o system, the memory system, and computer-computer communication structures.

**System Configurations**

We wanted to give the user considerable freedom

[1] Processor-Memory-Switch. The PMS notation is a scheme for concisely representing the "block-diagram" level of computer organization. Common abbreviations in PMS are P for processor, M for memory, S for Switch, K for control unit, C for computer. Abbreviations may be quantified by lower case letters such as c for central (i.e. Pc :=central processor), p for primary (i.e. Mp := primary memory), and s for secondary (i.e. Ms := secondary memory). For a complete description of the PMS notation see [3].

in specifying a system configuration with the ability to increase (or decrease) memory size, processing power, and external interfaces to people, other computers, and real time equipment. Overall, the PMS structure has remained essentially the same as the PDP-6 design, with periodic enhancements to provide more performance and better real time capability. (A PDP-6 memory or i/o device could be used on a KI10 processor, and a PDP-6 i/o device can be used on today's KL10 systems.) A radical change occurred with the KL20 to a more integrated, less costly, design for the processor, memory, and minicomputer i/o preprocessors.

The PMS block diagram of a two processor PDP-6 is given in Figure 3. But, for simple uniprocessor systems, the PMS structure was quite like our small computers with up to 16 modules on both the i/o and memory buses:

```
Mp...Mp          Kio-...KMs...
 |  |      Pc     |     |
-------          -------
(Memory Bus)     (I/O Bus)
```

Interestingly, a unified i/o-memory bus like the PDP-11 Unibus was considered. The concept was rejected since a unified bus designed to operate at memory speed would have been more costly.

The goal to provide arbitrary, modular computing resources led to a multiprocessor structure with shared memory. The interconnection between processors and memory modules was chosen to be a cross-point switch with each processor broadcasting to all memory modules.

An alternative interconnection scheme could have been a more complex, synchronous, message-oriented protocol on a single bus. More efficient cable utilization and higher bandwidth would have resulted but physical partitioning into multiple processor/memory subsystems for on-line maintenance would have been precluded. All in all, the crosspoint switch decision was basically sound although more expensive.

Figure 4 shows a PMS block diagram for the KA10 and KI10. There can be up to 16, 65 Kword, 4-port memory modules, giving a total of one megaword of memory. (Each processor addressed four Mwords.) With high-speed disk and tape units (e.g. 250 Kwords/sec.) a program controlled i/o scheme would place too much burden on the central processor. Therefore a direct port to memory was provided as in the PDP-6. In the KA10/KI10 systems, a switch (called a multiplexor) was introduced to expand the number of ports into memory to four for each Memory Bus used. The communications controllers were also expanded to handle more asynchronous and synchronous lines.

The KL10 was, by comparison, a radical departure from previous PMS structures (see Figure 5). In order to gain more performance, four words from four low order interleaved memory modules were accessed each cycle. The effective processor-memory bandwidth was thus over four Mwords/sec. The processor also connects to as many as four PDP-11 minicomputers (shown as C(11) in the figure). Most of the i/o is handled by these front-end computers.

Each PDP-11 can access the KL10 memory via indirect address pointers and transfers data in much the same manner as the peripheral processing units of a CDC 6600. Notice also that the KL10's console is tied to a PDP-11. This PDP-11 can load the KL10

Fig. 3. PMS diagram for PDP-6 system.

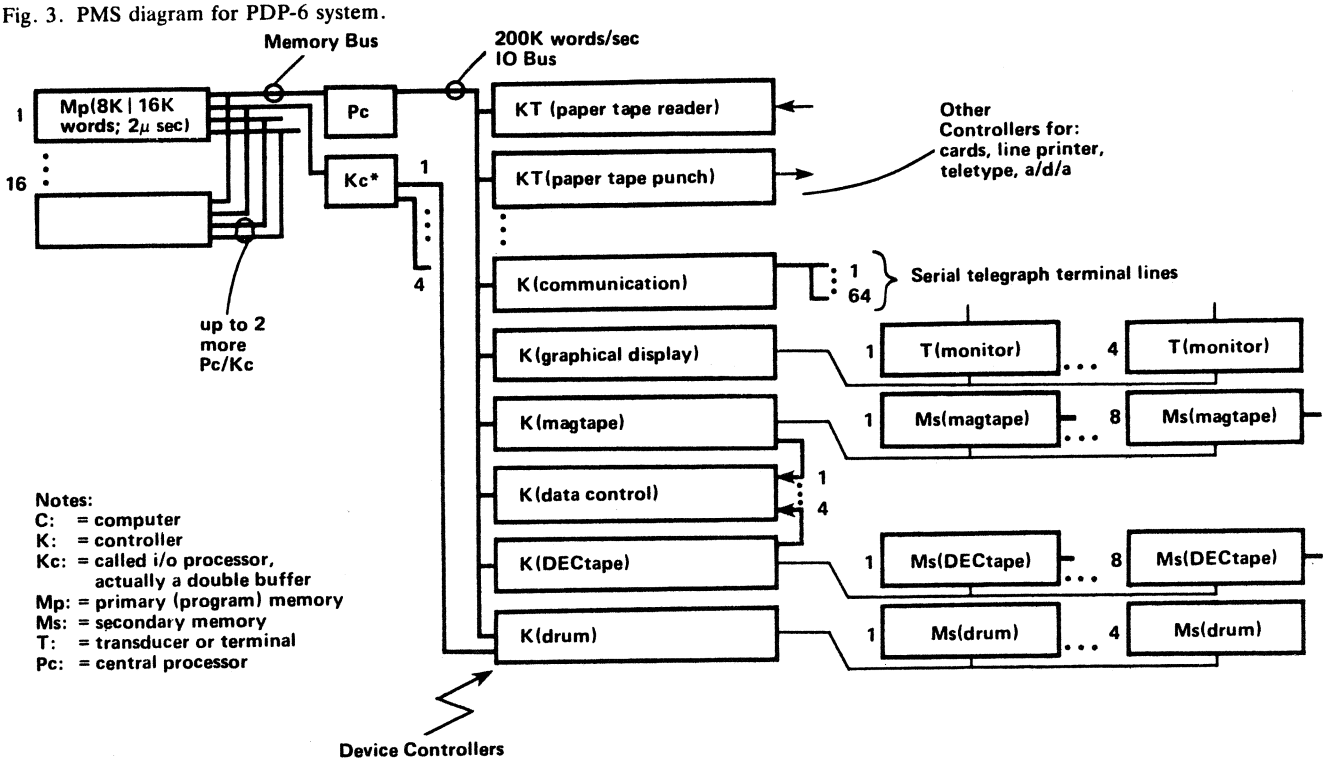


Fig. 4. PMS diagram for KA10 and KI10 processor-based system.

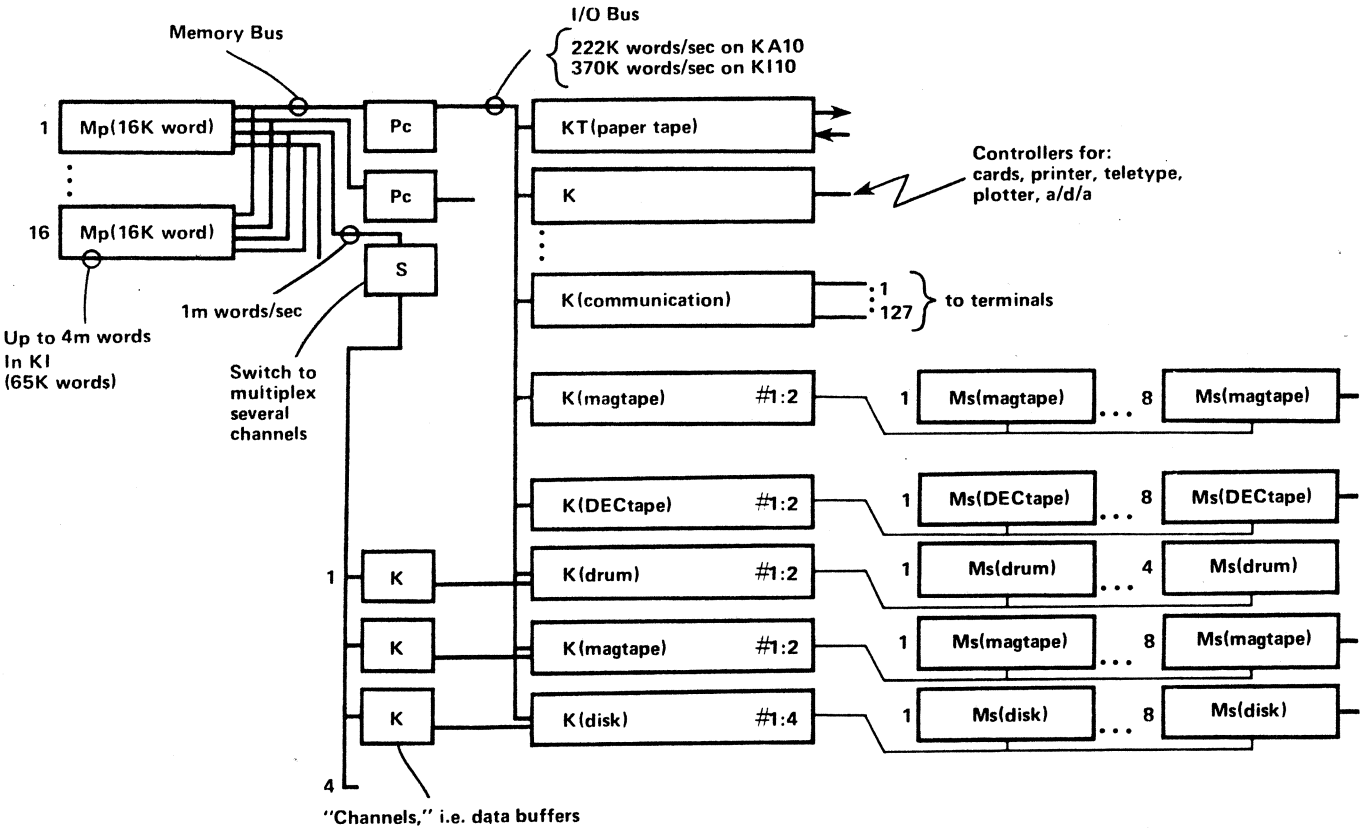
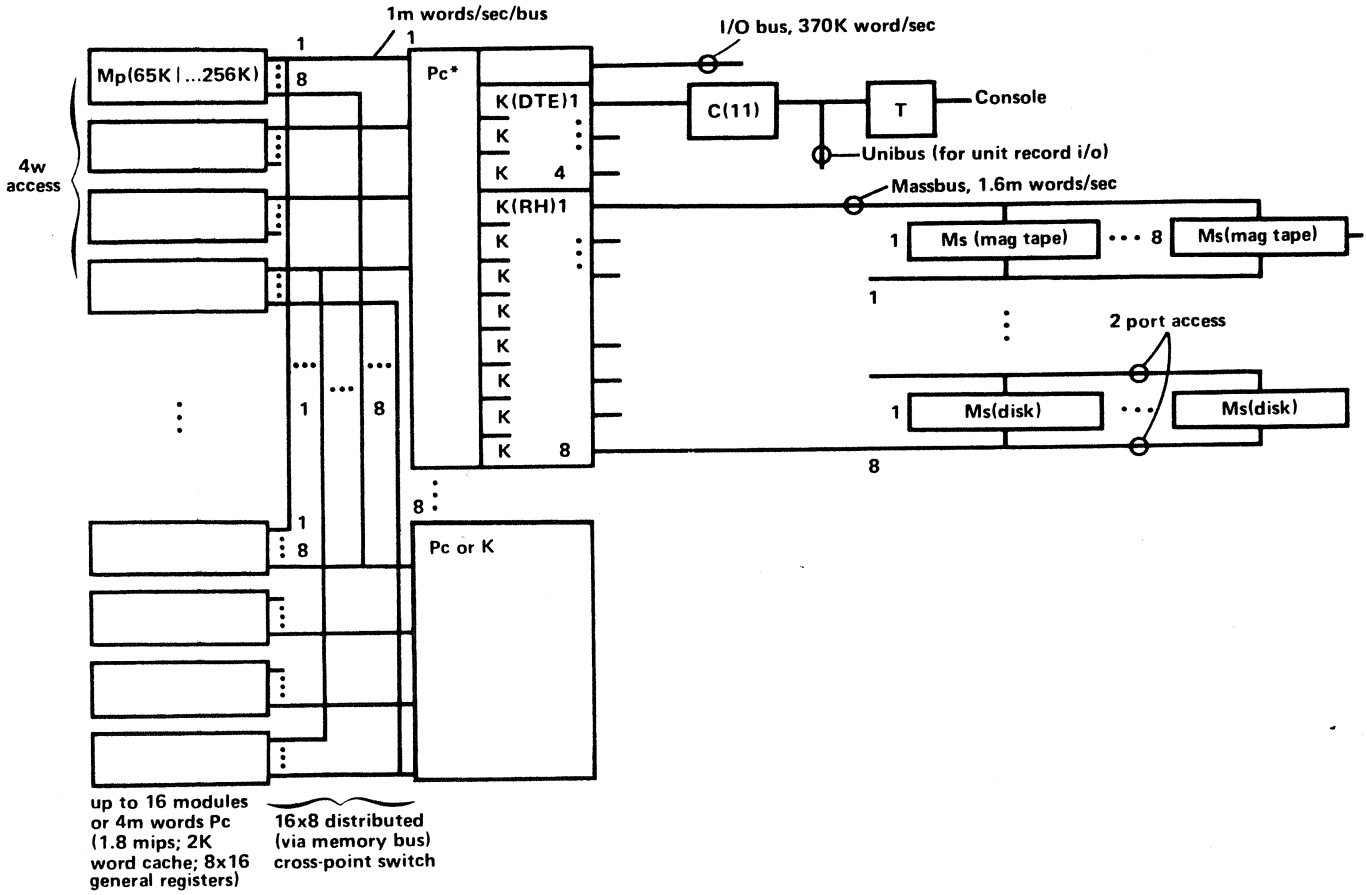



Fig. 5. PMS diagram for KL10 processor-based systems.



microprogram memory, run microdiagnostics, and provides a potential remotely operated console. Each of the PDP-11's can achieve a word rate of 70 Kchar/sec.

Up to eight DEC Massbus controllers are integrated into the processor. The Massbus is an 18-bit data width bus for block transfer oriented mass storage devices such as disks and magnetic tapes. Each Massbus can transfer 1.6 Mwords/sec. yielding a maximum 12.8 Mwords/sec. transfer rate for all channels. However, contemporary disks need about 250 Kwords/sec. so that all eight channels only require 2.0 Mwords/sec. of the 4 Mword/sec. memory bandwidth of 4 modules. Individual disks and tapes can be connected to a second port for increased concurrency. For larger memory configurations, a memory bandwidth of 16 Mwords/sec. is not uncommon. A 2 Kword processor cache provides roughly a 90% hit rate and reduces memory bandwidth demand by nearly a factor of ten.

The cost-reduced KL20 evolved by integrating the Massbus controllers and PDP-11 interfaces onto a single high-speed, synchronous bus. The model 2040 and 2050 computers are based on the KL10 processor and integrate 256 Kwords of memory in a single cabinet with the processor (thereby eliminating the external Memory Bus). The I/O Bus is also eliminated and all i/o transfers are either via the Massbusses or the PDP-11 i/o computers. (It must be noted that the 2040 structure is only possible because of the drastic increase in logic and memory density!)

**I/O System**

Relatively low-speed i/o (200 Kwords/sec.) in the PDP-6 was designed to be under central processor programmed control rather than via specialized i/o processors (IBM System 360/370 Channels). This method had proven effective in our minicomputers and was extended to handle higher data rates with lower overhead than specialized i/o processors.

The decision not to use the IBM-type channel structure was based both on high overhead (cost) in programming and hardware. Since i/o record transmission usually caused a central processor action, we felt the processor might as well transfer the data while it had access to it. This merely required a good interrupt and context switching mechanism, not another specialized processing entity. However, when an inordinately high fraction of the processor's time went to i/o processing, a second, fully general processor was added – not a processor that was fundamentally only capable of data transmission.

The PDP-6 interrupt scheme was based on our previous experience with a 16-level and 256-level interrupt mechanism for PDP-1. The PDP-1 scheme was an extension of the Lincoln Laboratory TX-2 [6]. The PDP-6 had a 7-channel interrupt system and each device on the I/O Bus could be programmed to a particular level. Hence a programmer could change the priority of a particular device that caused interrupts on the basis of need or urgency. The PDP-6 also had an i/o instruction (Block Input or Block Output) to transfer a single data item, between a block (vector) in primary memory and an i/o device. Thus as each word was assembled by a controller, an interrupt occurred and the block transfer was executed for one word, taking only three memory references (to the instruction, to increment the address pointer and block counter, and to transfer data). Most of the hardware to control the count and address pointer was already part of the processor logic.

In applications requiring higher data transmission (e.g. swapping drums, disks, TV cameras) a controller with a data buffer (erroneously called an i/o processor) and link to memory was provided. These controllers only required a single memory reference per data transfer with the address pointer and block counter in hardware. In the KA10 the name was changed to channel, and parameters for transferring contiguous records into various parts of memory were part of the channel's control. The device control was via the I/O Bus, and hence we ended up with a structure for high speed device control not unlike the IBM channels we originally wanted to avoid.

Competitive pressure from the Xerox Sigma series caused a change in the way interrupts were handled beginning with the KI10. Although the Xerox scheme had many priority levels, its main utility was derived from rapid dispatch to attend to a particular interrupt signal. We kept compatibility with the 7-channel interrupt by using a spare wire in the bus and adding the ability to directly dispatch to a particular program when a request occurred. At the interruption, the processor sent a signal to requesting devices and the highest priority device responded with a 33-bit command (3-bit function, 18-bit address, 12-bit data). The functions were: 1. Execute the instruction found at addressed location; 2. transfer a word to/from addressed location; 3. trap to addressed location; and 4. add data to addressed location. Little use was made of these functions (especially number four), since only a small number of devices were typically connected to a large system thus relaxing the requirement of rapid dispatch. Anyway, the competitive problem was solved (or went away as Xerox left the competitive scene). In systems that did have a large number of devices, a front end i/o processing minicomputer was more cost-effective than central processor controlled i/o.

**Memory System**

Because it was unclear how memory technology would affect memory speed, a completely asynchronous, interlocked memory bus was designed. Thus the 16 fast, general registers, the initial five microsecond memory, and the next generation two microsecond memory could all operate on a single system. (Most memories are now less than one microsecond cycle time.) The asynchronous bus avoided the problem of

Table V. Computer interconnection structures.

| Scheme | Data Rate | Structure | Models | Examples |
|---|---|---|---|---|
| Standard communication link | 110, 300 1200, 4800, 9600, 50K bits/sec. | network | all | |
| Special parallel, block transfer via hardware or software | 100K-1M words/sec. | tightly coupled | all | |
| Multiprocessors | at mem. access rate | multiprocessor | all | 2 Pc<br>16 Pc, proposed |
| Access into mini address space with interruption | at mem. access rate | multiprocessor shared memory | PDP-6 | The large computer accesses data in the small computer |
| The mini can transfer data into large machine via special control | at mem. access rate | tightly coupled | KA10-KL10 | Scheme used to interconnect minis to do i/o.<br>Multiple logical channels are provided |

distributing a single high-speed clock and allowed interleaved memory operation.

Modularity was also introduced to clarify organizational boundaries within the company and to make possible low cost, special purpose, production and engineering testers for the memory and i/o equipment. We believe the concept of well-defined modules was relatively unique, especially for memory, and was the basis for the formation of third party add-on memory vendors. MIT and Stanford University purchased memories from Fabritek and AMPEX respectively in the mid 1960's to start this trend. (Note, this design style differed from the IBM System/360 design with its relatively bounded configurations and integrated memory. Add-on memory did not appear until the early 70's for the IBM machines because, we believe, of the difficulty of the interface definition.)

The KI10 memory system was improved by assigning signals to request multiple, overlapped memory accesses and to increase the address size from 18 to 24 bits. The additional physical memory addresses are mapped into a program's 18-bit addresses via the core-held page table.

The KL10 processor-memory organization was a significant departure from the KI10 as previously discussed. The KL20 eliminated the original Memory Bus to provide an integrated system. It should be noted that this evolution was based on the drastic size reduction (a factor of about 300) from a single cabinet (6′ × 19″ × 25″ or about 34,000 cu. inches) for 16 Kwords to a single logic module for 16Kwords (15″ × 8″ × 1″ or about 120 cu. inches).

### PMS Structures for Computer-Computer Intercommunication

Throughout the evolution a number of schemes have been used to interconnect with other (usually smaller) computers. The schemes are given in Table V. Note that the first four schemes were conventional, while the last scheme was used in the KL10/20 structure so that an attached PDP-11 minicomputer could transmit data directly into the memory of the KL. This scheme was first used in the early 1970's for handling multiple communication lines.

## 6. Operating System

### PDP-6 Monitor Design Goals and Philosophy

The initial goals and constraints for the user environment are summarized in Table II. The most important goal was to provide a general-purpose timesharing system. The monitor was to allow the user to run in the mode most suited to his requirements, including interactive timesharing, real time, and batch. In timesharing there was no requirement for a human operator per se. Instead, the operator's console was a user terminal with special privileges. Real time programs had to be able to operate i/o directly, locked in core, and batch was to be provided as a special case of a terminal job.

Because of the modular expandability of the hardware structure, the software system had to be equally modular to facilitate varying system configurations and growth. The core resident timesharing monitor was only fixed at system generation (i.e. IBM's SYSGEN) time when software modules could be added to meet the system requirements. The core space required for monitor overhead had to be minimized. Thus job-specific functions were placed in the user area instead of in the monitor. The first 96 locations of each user job contained pertinent information concerning that job. A temporary area (stack) for monitor operations was also included. In this way, the monitor was not burdened with information for the inactive jobs. This structure permitted the entire job state to be moved easily.

Adequate protection was to be given each user from other nonmalicious users. However the user was not protected against himself because various user status information in the job area could be changed to affect his own job. Since common system resources were allocated upon demand and deadlocks could occur, the term "Gentlemen's Timesharing" was coined for the first monitor.

Table VI. Monitor functions evolution.

| Facility | PDP-6 (1964) | KA10 (1967) | KI10 (1972) | KL10 (1975) |
|---|---|---|---|---|
| Protection | one segment per user | two segments with shared program segment (required reentrant programs) | four modes for shared segments | virtual machine with shared segments |
| Program swapping | core shuffling | core shuffling; with swapping (via drum disk) | paging used for core management | demand paging (job need not be wholly resident to run) |
| Facilities allocator | devices assigned to users upon request (deadlocks possible ⇒ gentlemen's timesharing) | spooling of line printer & card reader | spooling of all devices | |
| Scheduler | round robin scheduler | scheduler to favor interactive jobs using multiple queues | fairness and swapping efficiency considerations | parameters for scheduling set by system mgr.; priority job classes and "pie-slice" schedule |
| User files | user files on DECtape, Cards, and Magnetic tape | significant enhancement of file function; on-line, random access disk-based files | improved file structure reliability, error recovery, protection and sharing; mountable structures | disk head movement optimization; |
| Command control program | simple (to implement) requiring little state | evolution to more powerful, easier to use command language | Common Command Language (CCL) | extensions to CCL |
| Batch | no real batch | remote & local single-stream batch | multiprogramming batch | improved multiprogramming batch |
| Terminal handling & communications | asynchronous task-to-task communications (for interactive terminals) as monitor module | synchronous communications for remote job and concentrator stations; "birth" of networks with simple topologies; ARPA network | synchronous communications in complex topologies; new protocol; IBM BISYNC for 2780 emulation/termination | DECnet* communications |
| Multiprocessing | – | dual processor support (master/slave) | high availability through bus switching hardware | symmetric multiprocessing |

* DECnet is DEC's computer network protocols and functions.

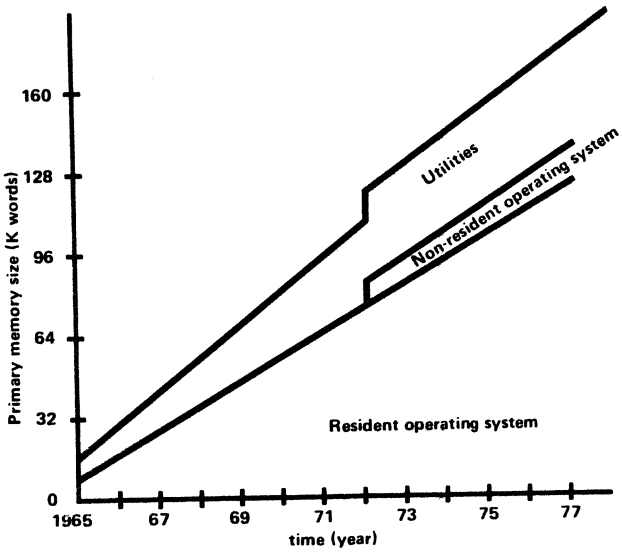
The UUO (Unimplemented User Operation), or system call instruction, provided both monitor-user communication and upward hardware compatibility. In the latter case, the instruction would use the hardware if available, otherwise the instruction would trap to the monitor for execution. For example, double precision hardware was available on later CPU models. The number of UUOs implemented in the monitor for monitor-user communication has been significant. The initial use of UUO's included requests for: core, i/o assignment, i/o transmission, file control, data and time, etc.

## PDP-6 Monitor

The *Monitor* was the name given to a collection of programs that were initially core resident and provided overall coordination and control of the operating environment. A nonresident part was later added with the advent of secondary program swapping and file memories (i.e. drum and disk). The Monitor did not include utilities, languages, and their run time support.

The PDP-6 Monitor was constrained to run in a 16 Kword (minimum) machine with console printer, paper tape reader (for maintenance) and two DECtape units. DECtape was a 128 word/block, block addressable medium of 450 Kcharacters for which a file system was developed. Memory minimizing led to very sparing use of shared tables. The key global variable data was

Fig. 6. Monitor and main utilities program size versus time.




restricted to: core allocation table, clock queue, job table, linked buffers for Teletype and other buffered i/o devices (e.g. DECtape directory), and a directory of system programs and monitor facilities.

The original PDP-6 Monitor was less than six Kwords. The monitor has increased at about 25%/

year with the KA10 at 30 Kwords, KI10 and 50 Kwords, and KL10 at 90 Kwords (see Figure 6). This increase provided increased functionality (e.g. better files, batch, automatic spooling), larger system configuration size, more i/o options, increased number of jobs, easier system generation, and increased reliability (e.g. checking, retries, file backup).

Note that with a 16 Kword memory, a nine Kword Fortran compiler with five Kword runtime package, and one Kword utility programs, two users could simultaneously reside in PDP-6 memory and use the machine for program creation and checkout. By keeping the monitor program size small, subsequent functionality increases kept the monitor module sizes in bounds such that program swapping was reduced. This provided high performance for a given configuration with little monitor overhead.

**Monitor Structure**

Table VI summarizes the development of the monitor with the various systems. The facilities are arranged beginning with basics. The following sections will deal with the various facilities, in turn. *Memory Protection Swapping*—the basic environment was discussed above in the ISP section on Multiprogramming/Monitor Facilities. *Facilities Allocator*—The Facilities Allocator was a module called from a console or program for an i/o device or memory space request. This module would attach (or assign) a given peripheral or contiguous physical memory area to a given job. Although this module was relatively trivial initially, it evolved to a more complex module since improper resources allocation caused deadlocks.

The KA10 generation software introduced queued operation. A line printer (output), paper tape (input/output) and a card reader (input) spooler were implemented. These spoolers ran as timeshared jobs, accepted requests from other user jobs and managed the input/output operation.

*Program Scheduler*—the scheduler was invoked by line frequency (50 or 60 Hz) interrupts to examine run queues and to determine the next action. The first monitor employed a round-robin scheduling algorithm. At the end of a given time quantum of 500 milliseconds, the next runnable job was run. A job was runnable if not stopped by the console and when not waiting for i/o.

Because terminal response time is the user's measure of system effectiveness, subsequent scheduler improvements have favored interactive jobs. With the KA10, separate priority queues were added so that jobs with substantial computation were placed in the lowest priority and then run the longest without interruption. This, in effect, approximated batched operation; for example, jobs from a card reader would operate as a batch stream. Later, batch operation was added for interactive users.

The introduction of disk/drum swapping caused additional complexities since runnable jobs might be located in secondary memory. The concept of "look ahead" scheduling was required and a more complex queueing mechanism was implemented. As the monitor selected the next job to be run, it would "look ahead" to determine future queues, and invoke the swapping module if required to move a runnable job into core. Because of the higher swapping overhead it was essential to run large jobs longer and less often. A "fairness" consideration also assured that each job, whatever its size, received enough run time to maintain responsiveness.

Recent enhancements permitted a Systems Manager to set scheduling parameters including established priorities of job classes. A "pie-slice" where classes of users are guaranteed fixed parts of the machine time and resources.

*User Files and I/O Device Independence*—in the initial PDP-6 design, resources such as magnetic tapes, unit record devices (e.g. card readers, line printer, paper tape reader/punch) and DECtapes (which were file structured) were requested by each user as they were required. The monitor allocated the device to a requesting given job until released.

I/O calls were evoked by the UUO call instructions. A particular device program call could specify the number of i/o buffers to provide so that arbitrary amounts of overlapped i/o and computing could be realized.

In order to realize the goal of modularity, each i/o device handler was implemented as a separate module. These modules used a common set of subroutines. The device tables were made as identical as possible to help achieve the device independent goal. Thus, a user specified an i/o channel, not a specific i/o device. The channel to name assignment could take place at various times from log-on to program run-time.

In the original monitor, a user was allowed to assign file devices to his job and read and write named files with the devices. Permanent, one-line user files with automatic backup were not implemented until the KA10 generation monitors. The concept of Project/Programmer Number was adopted (after MIT's CTSS) in order to provide increased file security and sharing. A user was required to enter a project/programmer number with his associated password. This not only established a job, but identified the user to the monitor. In addition to having resource privileges associated with better ID numbers, the user received a logical disk area for files. File access can be allowed (by the creator of the file) to any of the following levels with decreasing protection (increasing privileges): no access; execute only; plus read; plus append; plus update; plus write; plus rename; plus alter protection.

Significant evolution occurred in the user file facility. Improved file structure reliability and error recovery (such as writing pointer blocks twice) were achieved. With moving head disk availability, disk head movement optimization for file transfers on single or multiple drives was added. The concept of "mount-

able" structures was implemented to allow disk packs to be mounted and dismounted during timesharing operation as well as allowing a user to have a "private" pack mounted. As the number of users supported on the system and the diversity of their applications grew to include "business data processing" both hardware and software allowed expansion of the number and capacity of on-line disks.

*Command Control Program*—this program processes all commands addressed to the system from user terminals. Thus terminals served to communicate monitor commands to the system, to communicate to the user programs, and to serve as an i/o device for user programs. Terminal handling routines were an integral part of the PDP-6 Monitor. The original commands were designed to minimize the amount of state in the Monitor. As a result, users had to type several commands to control programs. A much more powerful command language evolved.

**Batch Processing**

Batch processing has evolved from the original, fully interactive PDP-6, where a user was expected to interactively provide commands for each step in the generation/execution of a program. The first batch on the KA10 was based on a user-built command file that mimicked his terminal actions. The user invoked this command file to execute his programs. Later, a multiprogrammed batch system was added and the job control syntax evolved to provide more functions per command. However, batch/interactive command commonality has been preserved through the current monitor versions. Still, batch control ran as a timeshared job using queued batch control files. Thus, the ability to log in a job, run to completion, and log off, is accomplished from a card reader, or any other storage or file device. Symbiant (queued) operation allowed control of card readers, line printers, etc., by the batch control program so that the machine could be scheduled more effectively. During this batch evolution, little monitor enhancement was necessary to specifically address the batch environment. Modules to improve efficiency (by multiple strands and better scheduling) and increase functionality were implemented as "user" jobs and interprocess queueing allowed communication between the "user" modules.

A line printer spooler, for example, was run as one of many jobs by the operator—a notion that evolved beginning with the KA10. If a special form was required for a print job, the operator would be notified and act accordingly. The user was relieved of this responsibility. Operator allocation, control, and media loading of the card reader, magnetic tape, private disk pack, DECtape, and plotter were provided in the KI10.

*Terminal Handling and Communications*—we believe the users' perception of system effectiveness related directly to his feeling that he was interacting and was in control. The requirement to communicate effectively with the user via the terminal was one of the most difficult design constraints. The very first version of the Monitor used half duplex communication for simplicity. But finally we decided to pay the additional price to gain the benefit of full duplex communication, i.e. being able to continuously input and output independent of system load. These philosophies have guided subsequent monitor generations.

A hardware module was constructed to facilitate terminal communication. This hardware was called the scanner because it looked at all the interface lines connected to Teletypes and interrupted the software when a character was received or needed to be transmitted. These line units, which we built on a single card, formed the basis of the UART (Universal Asynchronous Receiver Transmitter) LSI chip. A software monitor, called SCNSER (Scanner Service) handled interrupts from the hardware. SCNSER provided the important function of logically coupling a physical terminal with a job running under timesharing. The user was never burdened with attempting to relate his terminal with his job. This software module, by far the most logical complex part of the monitor has been rewritten two times to increase terminal functionality.

Later the KA10 terminal interface was implemented via a "front end" concentrator PDP-8 computer for large numbers of terminals—particularly where variable line speeds were involved (up to 300 baud). This implementation allowed some off-loading of the processor. Characters were assembled (serial parallel conversion) in the front-end PDP-8 and communicated with the KA10 via the I/O Bus on an interrupt basis.

In 1971 a front-end PDP-11 was provided direct memory access over the I/O Bus. This connection provided high speed, full-duplex, synchronous communications and was the prototype for the current KL10/PDP-11 front-end computer. Software modules were added to the Monitor to allow these synchronous lines to terminate remote PDP-8 and communication concentrator stations in simple point-to-point topologies. A remote station (e.g. line printer) is viewed by the user in the same manner as is a local printer.

With the KI10, a second front-end was produced which allowed BYSINC protocol of the IBM 2780 terminal to be used. However, most of our users were laboratory oriented and wanted greater performance and functionality. Thus, concentrator/remote station capability including route-through (i.e. communication via multiple concentrators) and multiple hosts was added. These formed the basis of some of our understanding for subsequent DECnet protocol standards and functions. The use of DECsystem 10 in the Advanced Research Projects Agency (ARPA) funded projects formed another key base for our DECnet protocols and functions [12].

DECnet 10 now provides the capability to have processes in different computers (including PDP-8's and PDP-11's) communicate with each other. These

jobs appear to each other as i/o devices in the simplest applications.

Throughout all of this communications functionality evolution, the goal has been to free the user from concern with the link, communications mode, hardware location, and protocol.

**Multiprocessing**

Although we predicated the original PDP-6 hardware on multiprocessing, the monitor was not designed explicitly for it. Lawrence Livermore Laboratory did build a two processor system with their own operating system and special segmentation hardware. To meet the needs of the predominately scientific/computation marketplace in achieving higher processor throughput, a dual-processor KA10 was implemented using a master/slave scheme with wholly shared memory and one monitor. The slave CPU scanned the queue of runnable jobs, selected one and ran it. If a monitor call was encountered, the job was placed in the appropriate queue and the monitor located another runnable job. The "master" handled all i/o and privileged operations. In a CPU-bound environment, the dual processor provided approximately a 70% increase in system throughput.

An off-shoot (and evolved design goal) of the dual processor implementation was higher availability. Monitor reconfigurability and bus switching hardware allowed redundant components to be fully utilized during normal operation and, in the case of a hardware malfunction, separated into an operating configuration (with all available i/o) and a maintenance configuration (consisting of CPU, memory, and the faulty component).

At Carnegie-Mellon University (CMU) we proposed to build a 16 to 32 PDP-10 structure [2]. It would have 16 Mwords of primary memory available via 16 ports at a bandwidth of 2.1 to 8.6 gigabits/sec. Using larger than KL10 processors, performance would have been over 50 mips (million instructions per second). The 16 processor, C.mmp [13] based on PDP-11's at CMU is a prototype of such a system.

**Languages and Utilities**

Monitor commands called the utilities and languages. The utilities, we called CUSP (for Common User System Program), and languages included: EDIT, an editor for creating and editing a file from a user console; PIP, the peripheral interchange program to convert information among the i/o media and files; LOADER to load object modules; DESK, an interactive calculator; MACRO, an assembler; and Fortran II. Figure 1 shows these programs at various times, together with their origins.

Utilities and languages have taken advantage of the interactive, terminal-oriented environment. Thus highly interactive editing/debugging facilities have evolved in terms of the program's own symbols. The file/data transfer utility, PIP, for Peripheral Interchange Program, is still in existence today, although in a much enhanced form. It has since been expanded to support the peripheral devices and the data formats encountered in the DECsystem-10 memory and i/o devices. Such a utility eliminated the need for a "library" of utilities and conversion programs to transfer data between devices. Such tasks as card-to-disk, card-to-tape, tape-to-disk, etc., conversion are controlled by a terminal using common PIP commands. PIP evolved in a somewhat ad hoc fashion from one or two Kword size in 1965 to ten Kwords with substantial generality.

A powerful and sophisticated text editor, TECO (Text Editor and Corrector) was initially implemented at MIT using a graphics display. TECO is character-string oriented and requires a minimal number of keystrokes to execute commands. It included the ability to define programs to do general string substitution. As the sophistication of users was later perceived to decline, the powerful editor created training and use problems. Thus a family of line- and character-oriented editors evolved which were easier to learn and remember. These were based on other line-oriented editors, but especially Stanford's SOS, which replaced the initial DEC line editor in 1970.

Many of the higher level languages were initially produced by non-DEC groups and made available through the DEC User Society (DECUS). For example, APL, Basic, DBMS and IQL (an interactive query language) were purchased from outside sources and are now standard, supported products.

BLISS, Basic Language for Implementing System Software, developed at Carnegie-Mellon University, became DEC's systems programming language [14]. A cross-compiler was subsequently developed for the PDP-11. Its' use as a systems program language has been due to the close coupling it provides to the machine, its general syntactic and block structures, and its high quality code generator. BLISS has been used for various diagnostic programs, the BLISS Compilers, the PDP-10 APL Interpreter, recent Fortran IV compilers for both PDP-10 and PDP-11, and the Basic +2 system. BLISS has also been used extensively within DEC for Computer Aided Design Programs.

**Tenex and the TOPS 20 Operating System**

Bolt, Beranek and Newman started a project in 1969 to build an advanced operating system called Tenex based on a modified KA10 (including rather elaborate paging hardware). This work was influenced by both the Berkley SDS 940 and MIT Multics Systems. Subsequently Tenex influenced and improved the KI10 design and became the base of TOPS 20. The system was described by Bobrow et al [4], and the three major goals stated in the reference were:

I. State of the Art Virtual Machine

a. Paged virtual address space equal to or greater

than the addressing capability of the processor with full provision for protection and sharing.

b. Multiple process capability in virtual machine with appropriate communication facilities.

c. File system integrated into virtual address space, built on multilevel symbolic directory structure with protection, and providing consistent access to all external i/o devices and data streams.

d. Extended instruction repertoire making available many common operations as single instructions.

II. Good Human Engineering Throughout Systems

a. An executive command language interpreter which provides direct access to a large variety of small, commonly used system functions, and access to and control over all other subsystems and user programs. Command language forms should be extremely versatile, adapting to the skill and experience of the user.

b. Terminal interface design should facilitate intimate interaction between program and user, provide extensive interrupt capability, and full ASCII character set.

c. Virtual machine functions should provide all necessary options, with reasonable default values simplifying common cases, and require no system-created objects to be placed in the user address space.

d. The system should encourage and facilitate cooperation among users as well as provide protection against undesired interaction.

III. The System must be Implementable, Maintainable, and Modifiable

a. Software must be modular with well defined interfaces and with provision for adding or changing modules clearly considered.

b. Software must be debuggable and reliable, allowing use of available debugging aids and including internal redundancy checks.

c. System should run efficiently, allow dynamic manual adjustment of service if desired, and allow extensive reconfiguration without reassembly.

d. System should contain instrumentation to clearly indicate performance."

Dan Murphy (one of Tenex's designers/implementers) came to DEC and led the architecture and development effort that produced TOPS 20. The effort at DEC has been to increase the performance of TOPS 20 to be competitive with the highly tuned Monitor while not losing its generality. The TOPS 20 structure does provide increased reliability and modifiability.

## 7. Hardware Implementation

While logic and memory technology are often the prime determinant of the performance and cost of a computer system, fabrication and packaging technology are equally important. This section surveys logic, fabrication, and packaging technology as it affected the various DECsystem 10 models. Table VII summarizes the various technologies.

### Logic

The PDP-6 used a set of logic modules that evolved from the earlier PDP-1, which in turn were derived from the Lincoln Laboratory circuits developed for the TX-0 [8] and TX-2 [6, 11] computers as part of the air defense program. These circuits were direct coupled transistor logic and included both series and parallel transistor circuits to give great flexibility in designs. The PDP-1 circuits operated at a 5 mHz clock, and new transistors enabled the PDP-6 circuits to operate at 10 mhz. The computer's clock was derived from a delay line which carried pulses generated by a pulse amplifier using pulse transformers (this too came from Lincoln Laboratory via the early work at MIT on radar and pulse transformers). The pulses were used for register transfer operations (i.e. moving data among the registers) and some logic gating.

Instead of using a small number of lines in a fixed, synchronous clock, many delay lines were used. The route through the control path determined the state of the machine. At each decision point, the next line or chain (set of lines) was selected. Hardware subroutines were also unique with this implementation. A control sequence consisting of a set of delay lines was defined as a subroutine and a calling module marked the calling site (e.g. add, subtract, and complement are at the lowest level). The basic multiply subroutine used add or subtract, and finally floating multiply used the normalize, and multiply subroutines. In this way, the implementation was kept structured and turned out to be quite straightforward. The flowcharts for the PDP-6 were only 11 pages, where each page has about 25 unique statements (actions), yielding a total of only 250 microsteps (each step causes 1 to 6 operations and corresponds roughly to current microprogram statements). The asynchronous adder was designed so that on completion of all the carries, the sequence would restart. Thus we took advantage of the observation made by von Neumann, et al in 1946, [3, ch. 4] that the average number of carries is $\log_2 36$ or slightly over 5, versus the worst case of 36. And since the average delay time was about 20nsec per carry, this reduced the average add time to only 100nsec versus 720nsec, yielding a very simple and fast circuit.

The KA10 used essentially the same circuitry but with significantly better packaging so that automatic wire wrap backpanels could be used. Note that in Table VII, the existence of certain semiconductors was the basis of new machines. The TTL/H series logic

appeared about 1969 and formed the basis of a machine (the KI10) with roughly the same power dissipation and physical size as a KA10, but with a factor of 2.2 more performance. In scientific applications requiring double precision computation, this performance differential is much greater. Ironically, the TTL/Schottky (TTL/S) series was first available in production quantities about the time of the delivery of the KI10. The KI10 design was started earlier and design options chosen so as to preclude the subsequent advances in speed, power, and density that the TTL/S gave.

The other important logic advances employed in the KI10 were the MSI register file and associative memory packages. The register file provided four sets of accumulators and thus decreased the context switching time. (This probably had a higher psychological than real value but was useful where special devices were operated on a high speed, real time basis.) The associative memory package permitted the construction of a 32 word associative memory to support a paged environment.

The KL10 provides almost a factor of five performance improvement over the KA10 for programs using the basic instruction set. An even larger performance improvement is realized for Cobol or extended precision scientific programs. The organization and much of the base work for the KL10 was done by Dave Poole, Phil Petit, John Holloway, and Jack Wright at the Stanford Artificial Intelligence Laboratory.

The KL10 is microprogrammed using a memory based on the one Kbit bipolar RAM. A cache memory is also constructed from the one Kbit chips. The KL10 is implemented in the Emitter Coupled Logic (ECL) 10K series rather than the TTL/Schottky of the original Stanford design. It was felt that the ECL speed advantage with 3 nsec. gate delay vs 7 nsec. for Schottky was worth the extra design effort especially since the

Table VII. Implementations for DECsystem 10 hardware.

| Processor | PDP-6 | KA10 | KI10 | KL10 |
|---|---|---|---|---|
| Design start | 3/63 | 1/66 | 12/69 | 1/72 |
| First ship | 6/64 | 9/67 | 5/72 | 6/75 |
| Logic | Germanium, Silicon transistors | Discrete Silicon transistors and diode | TTL/H (MSI) Registers; assoc. memory | ECL 10K; Fast, 1 Kbit memories |
| MIPS(avg.) | 0.25 | 0.38 | 0.72 | 1.8 |
| Packaging (slice of Pc) | 1-bit of AR, MB, MQ, AD:88 transistors, 2-sided PC etch; 2, 18-pin & 2-22-pin conn. (11" × 9" boards) | implemented in R, S, W-series flip chip (discrete) modules (5 1/2 × 5 1/4 boards) | implemented in R, S, W, M-series flip chip (discrete + MSI) modules 5 1/2 × 5 1/4 boards | 6-bits of AR, ARX, MQ, BR, BRX, AD, ADX:70 MSI ECL per module; 216 pin connector; (8" × 16" boards) |
| Pc. size | 2 bay | 2 bays | 2+ bays | 1/2 bay (including internal channels) |
| Pc. price | $120K | $150K | $200K | $250K |
| Control Design | async. & subroutine logic | same as PDP-6 | clocked sync. | KL20 is clocked sync.; microprogrammed |
| Module Size | large modules | small modules wire wrap | same | large modules (16 Kword core memory module) |
| Registers | 16 | 16 | 4 × 16 | 8 × 16 |
| I/O calls | prog. interrupts UUO traps; | same | vectored interrupts | |
| I/O transmission | I/O & Memory Bus | added channels | | integrated controller for MASSBUS; I/O via PDP-11 computers |
| Memory Management | 18-bit phys. addr. protection & relocation regs. | 2 protection & relocation regs. for shared program segments | 22-bit phys. addr; paged using 32 word associative memory | 22-bit phys addr. paged, using associative memory via cache |
| ISP | see Table III (integers, floating) | conversion to assist d.p. float | hardware d.p. float | string & conversion for d.p. integers |
| Parallelism | | simpler (faster) data path | instruction look-ahead (4-word) fetch | instruction look ahead; 2 Kword cache memory |
| Fabrication | (too) large modules | Gardner-Denver automatic wire wrap for backpanel interconnection | semiautomatic wirewrap for twisted pair | large (hex) modules with many pins; low cost minis front end; (KL20) integrating Pc and Mp together – eliminating Memory Bus⇒ high density core memory modules |
| Consequences | served as PDP-10 production prototype | buildable in production | more performance (scientific & real time); and paging for operating systems | more performance via cache; microprogramming for better COBOL ISP; i/o computers; lower cost |

ECL needed more power and care to layout the board and backplane.

**Fabrication**

The Gardner-Denver automatic wirewrap machine was significant in the fabrication of machines. Automatic wirewrap economically provided accurately wired backpanels. As a more important side effect, it made the high volume, low cost fabrication of minicomputers possible! Some backpanel wiring on the KI10 and KL10 processors using twisted pairs cannot be done using the Gardner-Denver machinery. For this, DEC developed a semiautomatic wirewrap machine which locates the pins, and selects the wire length for an operator.

Computer design aids have evolved to support computer implementations on an "as needed" basis, barely keeping ahead of the implementations. These have included printed circuit board layout/routing, backplane layout/routing, circuit/logic simulation, wire length/logic delay checking, and various manufacturing aids. One notable exception to this trend has been the Stanford University Drawing System (SUDS) developed by the Stanford Artificial Intelligence Laboratory. SUDS was used for drawing the entire KL10 design. The design time and cost would have been significantly greater if SUDS had not been available.

**Packaging**

Semiconductor density is a major determinant of the system size, and size in turn is a major determinant of speed (e.g. shorter interconnection paths). Seymour Cray has stated in a lecture at Lawrence Livermore Laboratory (Dec., 1974) that for each generation of his large computers, the density has improved by a factor of five.

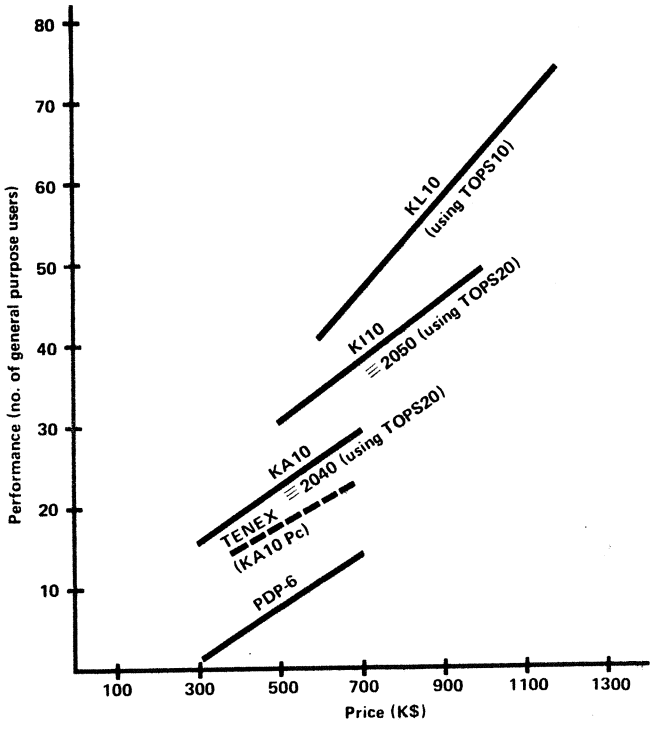
The packaging for the PDP-6 was identical to that of the PDP-1, 4, and 5 and used a board area of about 40 sq. in. with a 22 pin connector. A logic density improvement of two was achieved over the previous designs by using six special function modules. However this density turned out to be too high for the number of pins. A natural extension was a board twice as large with 44 pins. The most interesting module was the bit slice of the working registers: accumulators, multiplier-quotient, and memory buffer. This module required more than 44 pins, so the extra signals were bused across the back of the module. This busing increased module swap time and the mechanical coupling increased the probability that fixing one fault would cause another. Because of this, the designers of the KA10 and KI10 became fearful of large boards. Only with the KL10 in 1972 were large boards reintroduced into the DECsystem 10. On the other hand, large boards had been used in DEC minicomputers since 1969. Multilayered boards were required for the KL10 ECL logic. These boards were adapted from the multilayered boards developed for the TTL/S PDP-11/45 (1972).

**Price/Performance**

Surprisingly, over time the various models of the DECsystem 10 have been implemented at an essentially constant cost. The option to apply technology at constant performance with reduced price was never examined as an alternative strategy. In the minicomputer part of the company, both alternatives were vigorously pursued in order to provide a growing business and stimulate design alternatives. The relatively static DECsystem 10 strategy with constant price, no doubt, stems from the highly coupled interaction of: builders (wanting to go on to provide the next highest level of performance which was the founding principle of the group); the salespeople (many of whom came from other companies and are only used to working with a particular user class); users (who want more performance so as to reduce their overall cost/performance ratio); and marketing (which integrates needs and alternatives). This is illustrated in Figure 7. Here we give the performance in terms of the number of general purpose users versus the system price.

Figure 8 gives a single price of the system for each generation, together with the percentages going of each for the system components. The best cost/performance systems are shown (except, in the case of the minimal PDP-6). Figure 9 gives the price of the various processors versus time for the family; note the processor price has been increasing roughly at the inflation rate, suggesting a manpower intensive (or service-type) market structure. Note that since the performance (Table VII) has improved at roughly a factor of 10 in 10 years, the increase in performance/cost is nearly

Fig. 7. Performance (in general purpose users) versus price for each generation.



20% per year. In contrast, a minicomputer line (constant performance) is plotted which shows the price decreasing at 21% per year, with a factor of 10 price decline in 10 years. We should ask: could a PDP-6 level processor be built in 1975 to sell for $10K? Clearly it could, and such a system has been built as an advanced development project. This small 10 has a unified bus structure like the PDP-11 with a connection to use the Unibus family i/o devices. A system with 512 Kwords and the performance of greater than a KA10 occupies a cabinet somewhat smaller than an 11/70 minicomputer.

Figure 10 shows how the price of memory has decreased with time. Note that even though there was growth in the memory size of the monitor of 25%/year, there was a positive improvement in the memory price performance. In reality, many functions which the user was explicitly responsible for were moved to the monitor as basic operations. A similar plot for secondary memory prices is given in Figure 11.

Fig. 8. System component prices versus generation.

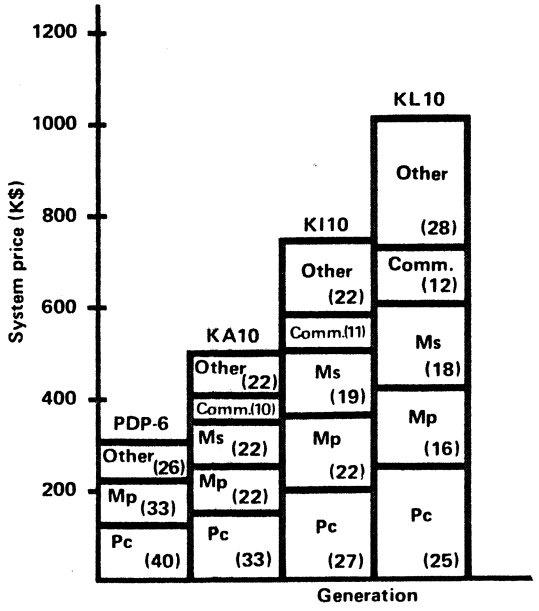
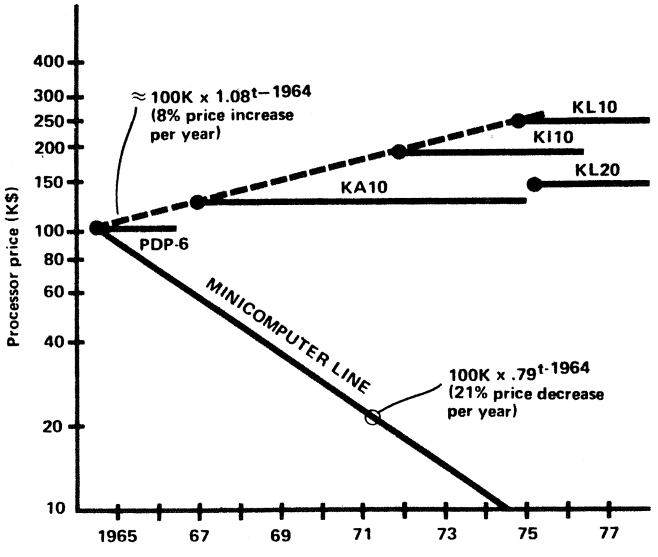



Fig. 9. DECsystem 10 processor price versus time.




## Conclusions

We believe the existence of the DECsystem 10 has been beneficial to the many environments for which it has provided real time and interactive computation, including the computer science and computer engineering communities. In turn, we have tried to respond to the needs of these users. Its existence has also been a positive force in encouraging alternative, competitive products in what otherwise might have been a dull, batch environment. The system has also been used by and influenced minicomputer, and now microcomputer development including: hardware technology (e.g. wirewrap); support for machine development (including simulation); and examplary design leading to time-sharing systems (e.g. DEC's TSS/8, RSTS) and user environments (e.g. RT-11 and microcomputer systems).

Fig. 10. DECsystem 10 primary memory price per word versus time.

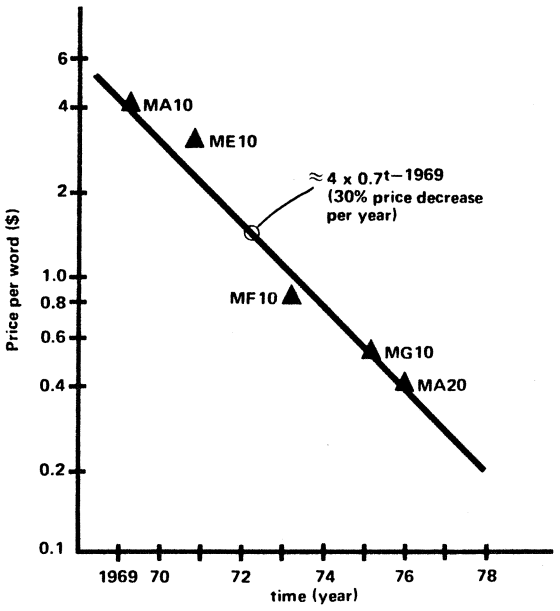
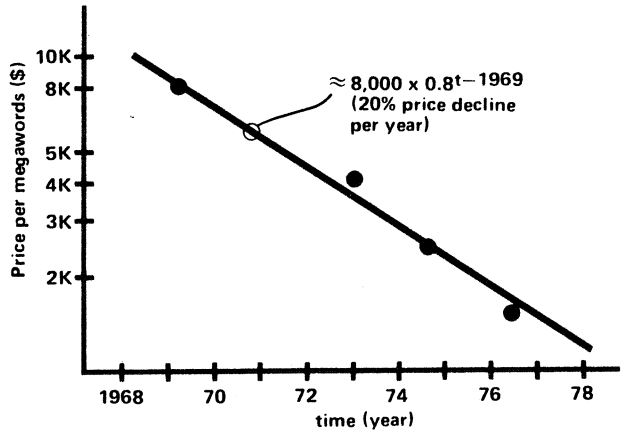



Fig. 11. DECsystem 10 secondary memory price per Mwords versus time.



We believe the key to the 10's longevity is its basically simple, clean structure with adequately large (one Mbyte) address space that allows users to get work done. In this way, it has evolved easily with use and with technology. An equally significant factor in its success is a single operating system environment enabling user program sharing among all machines. The machine has thus attracted users who have built significant languages and applications in a variety of environments. These user-developers are thus the dominant system architects-implementors.

In retrospect, the machine turned out to be larger and further from a minicomputer than we expected. As such it could easily have died or destroyed the tiny DEC organization that started it. We hope that this paper has provided insight into the interactions of its development.

*Acknowledgments.* Dan Siewiorek deserves our greatest thanks for helping with a complete editing of the text. The referees and editors have been especially helpful. The important program contributions by users are too numerous for us to give by name but here are most of them: APL, Basic, BLISS, DDT, LISP, Pascal, Simula, SOS, TECO, and Tenex. Likewise, there have been so many contributions to the 10's architecture and implementations within DEC and throughout the user community that we dare not give what would be a partial list.

Received April 1977; revised September 1977

**References**

**1.** Bell, G., Cady, R., McFarland, H., Delagi, B., O'Laughlin, J., and Noonan, R. A new architecture for minicomputers—the DEC PDP-11. Proc. AFIPS 1970 SJCC, Vol. 36, AFIPS Press, Montvale, N.J., pp. 657-675.

**2.** Bell, G., and Freeman, P. Cai—A computer architecture for AI research AFIPS Conf. Proc. Vol. 38 (Spring, 1971), 779-790.

**3.** Bell, G., and Newell, A. *Computer Structures: Readings and Examples.* McGraw-Hill, New York, 1971.

**4.** Bobrow, D.G., Burchfiel, J.D., Murphy, D. L., and Tomlinson, R.S. TENEX, A Paged Time Sharing System for the PDP-10. *Comm. ACM 15*, 3 (March 1972), 135-143.

**5.** Bullman, D.M. Editor, stack computers issue. *Computer* 10, 5 (May 1977), 14-52.

**6.** Clark, W.A. The Lincoln TX-2 computer. Proc. WJCC 1957, Vol. 11, pp. 143-171.

**7.** Lunde, A. Empirical evaluation of some features of Instruction Set Processor architecture. *Comm. ACM 20*, 3 (March 1977), 143-152.

**8.** Mitchell, J.L., and Olsen, K.H. TX-0, a transistor computer. Proc. EJCC 1956, Vol. 10, pp. 93-100.

**9.** McCarthy, J. *Time Sharing Computer Systems, Management and the Computer of the Future* M. Greenberger, Ed., M.I.T. Press, Cambridge, Mass., 1962, pp. 221-236.

**10.** Murphy, D.L. Storage organization and management in TENEX. Proc. AFIPS 1972 FJCC, Vol. 41, Pt. I, AFIPS Press, Montvale, N.J., pp. 23-32.

**11.** Olsen, K.H. Transistor circuitry in the Lincoln TX-2. Proc. WJCC 1957, Vol. 11, pp. 167-171.

**12.** Roberts, L.G. Ed. Section on Resource Sharing Computer Networks. AFIPS 1970 SJCC, Vol. 36, AFIPS Press, Montvale, N.J., pp. 543-598.

**13.** Wulf, W., and Bell, G. C.mmp—A mutli-mini-processor. Proc. AFIPS 1972 FJCC, Vol. 41, AFIPS Press, Montvale, N.J., pp. 765-777.

**14.** Wulf, W., Russell, D., and Habermann, A.N. BLISS: A language for systems programming. *Comm. ACM 14*, 12 (Dec. 1971), 780-790.