digital INTEROFFICE MEMORANDUM

TO: DISTRIBUTION

COMPANY PRIVATE

DATE: February 13, 1975

FROM: Dave Nelson

DEPT: 11 Planning/Architecture

EXT: 4509 LOC: ML5/E67

SUBJ: PROPOSAL FOR PDP-11 I/O ARCHITECTURE

I propose that the attached specification be adopted in future implementations of the PDP-11 architecture.

For background information, reference is made to Section I (I/O Systems) and Section II (Interrupt System) of the 1/22/75 version of the PDP-11 Architectural Enhancement Strategy publication.

Many discussions have been held with people in Software Engineering, Peripheral Engineering, Micro Products, Communications Engineering, Hardware Engineering, and R/D, all of whom have contributed in various ways.

The attached document appears somewhat revolutionary in that some new terminology is introduced. I believe, however, that this terminology is necessary if we wish to realize the full potential of the approach and avoid some of the past mistakes.

I'd appreciate whatever comments you have, and I'll be holding some design review sessions shortly.

Regards.

DN:elb

DISTRIBUTION

Phil Arnold
Jega Arulpragasam
Vince Bastiani
Gordon Bell ✓
Jim Bell
Ron Brender
Jack Burness
Roger Cady
Dick Clayton
Skip Coombe
Dave Cutler
Bruce Delagi
Bill Demmer

Tom Fava
Robin Frith
Lorrin Gale
Mike Garry
Andy Goldstein
John Holman
John Hughes
Chuck Kaman
Julius Marcus
Craig Mudge
Clay Neil
Jim O'Loughlin
Ralph Platz

Larry Portner
Bob Puffer
Maurice Richeson
Al Ryder
Grant Saviers
Bill Strecker
Steve Teicher
Nate Teichholtz
Mike Tomasic
Pete van Roekens
Larry Wade
Stu Wecker
Garth Wolfendale

PDP-11 I/O PHILOSOPHY

Some discussion regarding the philosophy and intended direction of the PDP-11 is warranted as an introduction to the proposed concepts and their motivations.

With the introduction of the 11/70 and 11/Q machines, the PDP-11 architecture will span a greater performance range than any other computer, with the possible exception of the IBM 360. However, this performance range is primarily restricted to the instruction set architecture. The I/O architecture is so directly coupled to the physical existence of the UNIBUS, both extremes of the performance range have had to make compromises. It is therefore a goal of this proposal to introduce concepts into the PDP-11 I/O structure that:

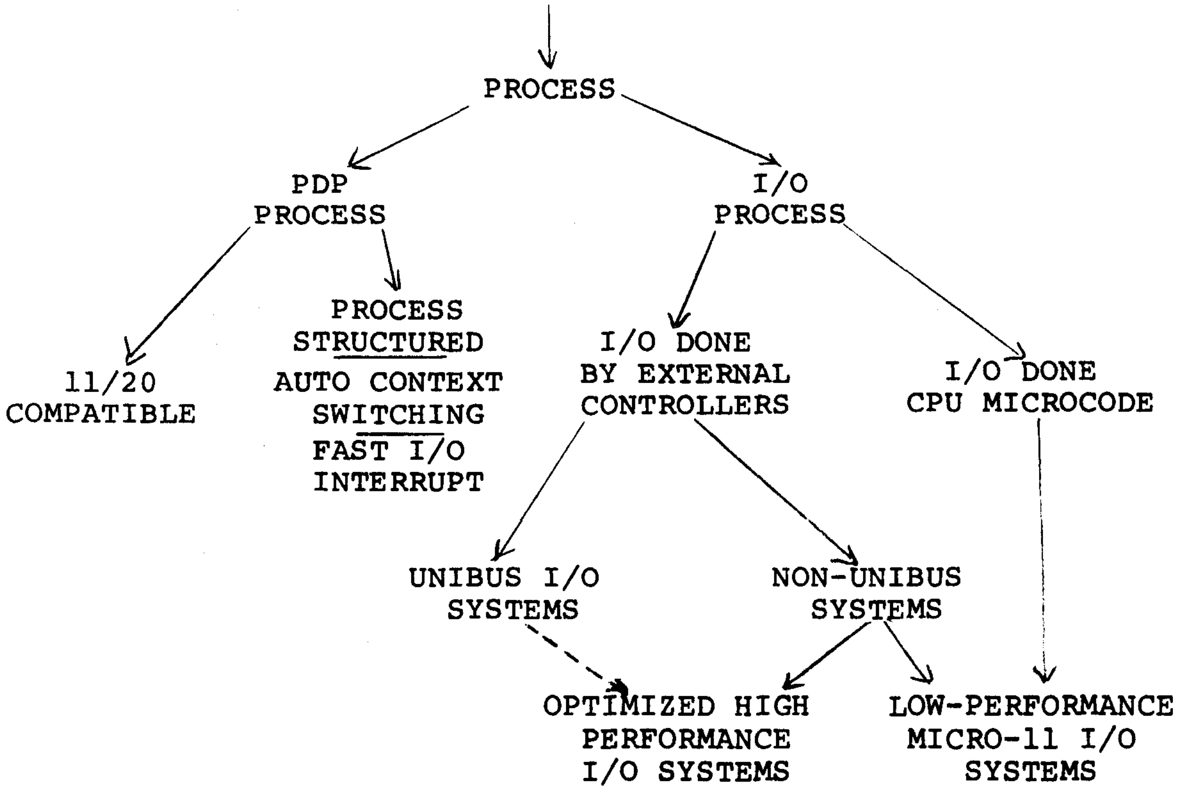
1. rely, to a lesser degree, on the UNIBUS; and

2. are amenable to a broader performance range for

   (a) low-performance I/O for micro-11s

   (b) high-performance I/O for large 11s

The following diagram intuitively characterizes some of the intents of this proposal; namely, to derive a machine structure which is conducive to all possible paths shown.



MOTIVATION

This proposal intends to provide a consistent and comprehensive approach to the following areas:

1. Fast Machine Context Switching

   A mechanism is provided that rapidly saves and restores machine registers when a process is switched.

2. Higher I/O Performance

   The architecture is more conducive to designing self-optimizing I/O systems that increase high-end performance and better manage UNIBUS bandwidth.

3. Low I/O Capability

   The architecture will allow us to develop low cost I/O systems for NON-UNIBUS low end 11s.

4. CPU Microcode I/O

   Higher performance at lower cost is attainable for communications and intermediate speed peripherals, by utilizing central processing resources for I/O functions.

5. I/O Page and Trap Vector Space

   The I/O page, now fully occupied, will contain descriptor registers which "float" in the address space. Device register information will be accessed through memory locations, and the trap vectors will be program assigned.

OVERVIEW

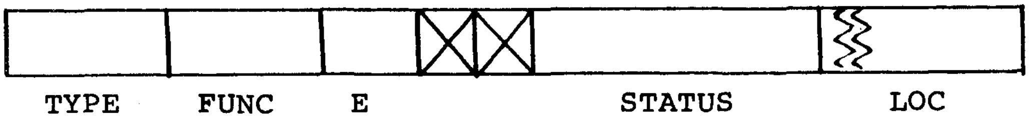
The concept of a process is introduced to describe both PDP-11 programs and I/O functions. Processes are uniquely defined by 32 bit process descriptors which, in turn, are located in memory space by 16 bit process numbers.

Processes are defined to be procedures with context which serves to distinguish between the descriptive information (context) needed to characterize the process, and the procedural information (actual mechanism, such as CPU, microcode, controller, I/O processor, etc.) required to carry out the process. Specifically, with regard to I/O, the architecture requires software to establish only the descriptive information, and thus is able to accomodate wide variants in the actual mechanisms and implementation techniques that perform the I/O function.

In terms of current PDP-11 definitions, the process descriptor is a generalization of the Program Status word and Program Counter. The process number for interrupt service routines is a generalization of the trap vector address. PDP-11 context is the general registers, KT map, etc., and I/O context is contained in the CSRs located in the I/O page.

PROCESS DESCRIPTOR FORMAT

The 32 bit Process Descriptor has the following format:



TYPE (2 bits) - Process Type

```
00,01,10 - PDP-11 process (Kernel, User, Spvr.)
11       - I/O process
```

FUNC (3 bits) - Function

-- previous mode and register select for PDP-11 processes

-- function code for I/O processes

E (1 bit) - Extended Context

-- used to select interpretations of LOC

STATUS (8 bits)

-- priority level and condition codes for PDP-11 processes

-- word count for non-extended I/O processes

-- status bits for extended I/O processes

LOC (16 bits) - Address Location

-- Program counter (virtual Kernel space) for non-extended PDP-11 processes

-- Location of context block for extended PDP-11 processes

-- Buffer Address (physical) for non-extended I/O processes

-- Location of context block for extended I/O processes

CONTEXT BLOCKS

For extended PDP-11 processes, the context block has the format:

(PREVIOUS MAP)	- address of previous KT-11 register image
(CURRENT MAP)	- address of current KT-11 register image (if zero, auto loading is suppressed)
R0-R6	- general registers
PC	- program counter (virtual Kernel space)

For extended I/O processes, the context block has the format:

PLINK	- process number of next process (normally the trap vector address)
DEVCTX	- device context (word count, buffer address, etc.)

PROCESS NUMBERS

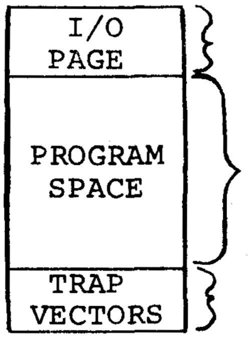
Since Process Descriptors reside in memory address space, one can consider this space an enormous Process Descriptor Table, whereby the location of all descriptors is specified by a 16 bit index called the process number. Processes are invoked by three distinct mechanisms:

1. An I/O process Descriptor, located in the I/O page, is invoked by CPU setting register bits. The process number is the address of device register (PD).

2. A PDP-11 or an I/O Process Descriptor, located in low core (trap vector space), is invoked by an interrupt request via the UNIBUS. The process number is the trap vector address.

3. A PDP-11 process (possibly an I/O process on future machines), located in program area, is invoked at software level by execution of an RTI. The process number is in R6, and the process descriptor is on top of the stack.

The location of process descriptors is therefore correlated to the mechanism that carries out the process. For example, I/O processes can either be performed by external controllers (in which case they reside in I/O page), or they can be performed by CPU microcode (in which case they reside in trap vector address space).



I/O Process Descriptors for externally controlled processes

PDP Process Descriptors for software level processes

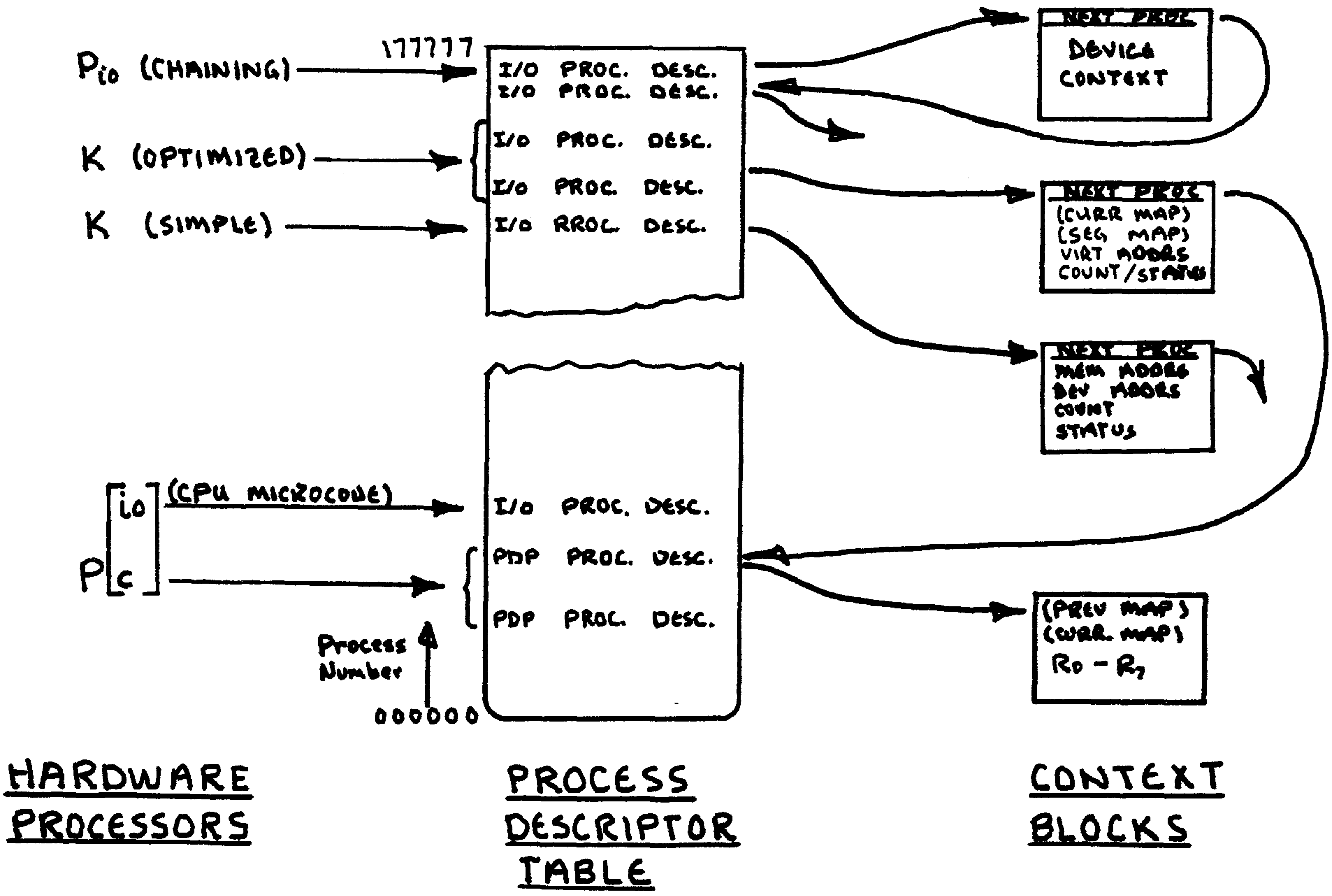
PDP and I/O Process Descriptors for CPU controlled processes

LINKED PROCESSES

In general, processes are linked together in a series of lists whereby processes in any given list are performed serially. This ensures orthogonality among processes that comprise a list, and serves to resolve data contention and synchronization problems discussed later. The list structure only applies to I/O processes and is characterized by the following:

-- for extended context processes, the first location in the context block contains the process number for the following process.

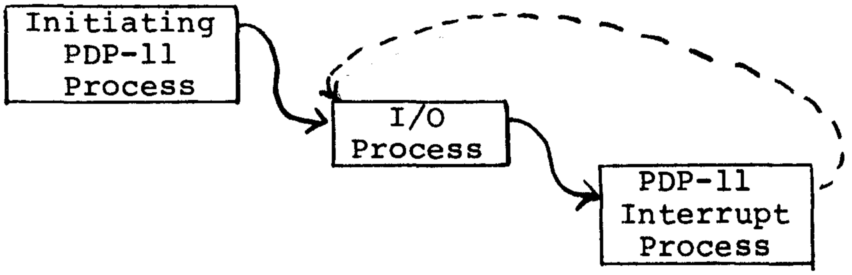
-- for non-extended context processes, the process number for the following process is the current process plus four (4 bytes).



Relation Between Hardware Processors and Descriptive Context Blocks

I/O PROCESS FOR AN EXTERNAL DEVICE/CONTROLLER

Process Descriptors for external devices reside in the I/O page. For the extended context case (normal for external devices), the descriptor contains the address of the context block which contains the device registers (address, count, etc.) and the next (linked) process number. The linked process number will normally be the trap vector address which locates the process descriptor (Ps, Pc) of the interrupt service routine.



In this specific case, the initiating PDP-11 routine would invoke the I/O process by initiating the process descriptor located in the I/O page (similar to current approach). The I/O process, once initiated, continues until the I/O is terminated, at which point it asserts the process number (trap address) of the interrupt service routine on the CPU. It should be noted that a similar I/O process descriptor could reside down in the trap vector space and operate under control of CPU microcode in a manner equivalent to I/O processes operating under control of external device controllers.

I/O PROCESS CONTROLLED BY CPU MICROCODE

Many block transfer devices, such as a floppy disk, may have controllers which, due to cost, are designed to interrupt on a per-word basis, each interrupt performing the data storage and memory increment. Other devices, primarily communication devices, are required to interrupt on a per-character basis in order to manage message protocols, control characters, buffer management, etc.

The current practice of interrupting the CPU for these relatively trivial reasons causes unacceptable burdens on the CPU and, consequently, unacceptable levels of performance. It is the purpose of this proposal to define the architectural relationships between those functions that can be done more efficiently by CPU microcode and those functions which require PDP-11 interrupt service routines.

The process-structured architecture discussed herein provides an integrated approach to all I/O processes, regardless of whether they are implemented by CPU microcode or an external controller. Further, the descriptors for all CPU functions are of the same form, regardless of whether they are PDP-11 processes or I/O functions.

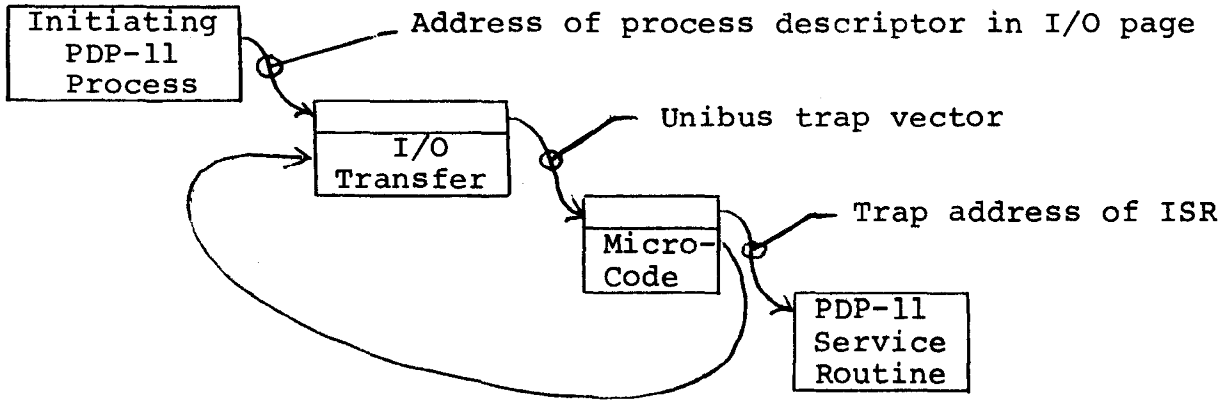
For CPU-controlled I/O functions, we will, in general, have the following events:

1. Initiating PDP-11 routine starts the device.

2. The device transfers data (1 word or character) and generates an interrupt request to the CPU.

3. The CPU handles the request in microcode logic, stores the data in memory, and determines if a PDP-11 service routine is required (word count expires, special character, etc.).

4. The CPU executes a PDP-11 interrupt service routine when conditions require.



PDP-11 AUTO CONTEXT SWITCH

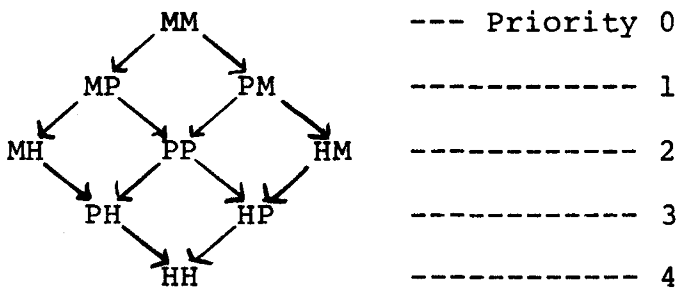
PDP-11 type process descriptors can be in the extended context format in which the low order 16 bits contains the virtual address in Kernel space of the processes context block. Context switches for these processes are performed automatically by the CPU, and can be invoked by either (1) the assertion of an interrupt by a device, or (2) the execution of an RTI instruction (used for task scheduling, as well as interrupt dismissal).

PDP-11 context consists of all eight general registers and the set of KT-11 map registers whose address is located in the context block (auto loading of map registers can be suppressed by specifying zero as the map address).

When a context switch is invoked, the current context is saved in the current processes' context block, and the new context is loaded from the new processes' context block. This deviates somewhat from the "stack" philosophy of saving current context on the new processes stack, having the disadvantage of providing maximum stack space for the maximum level of nesting.

The current proposal, however, requires that it be impossible for an extended process (one which has a context block) to interrupt a non-extended process, since the new process would load registers with new values without having saved the old values. Consequently, the rules for determining which processes can take advantage of auto context switching must be in accordance with the task's priority.

The following diagram generalizes the relation between priority level and the context switching mechanism. Here, M denotes switching by microcode, P denotes switching by program (usually saving only a part of the registers), and H denotes selecting a different set of hardware registers. The first letter in the pair is for the general registers, and the second letter is for the map registers. At any given time, the complete system must correspond to a single path connecting MM (both microcode) to HH (both hardware re-select).



These constraints satisfy the general intention to (1) auto switch tasks running at low priority levels, (2) partially switch intermediate-level tasks under program control, and (3) automatically switch hardware register sets for high-priority tasks.

COMPATIBILITY

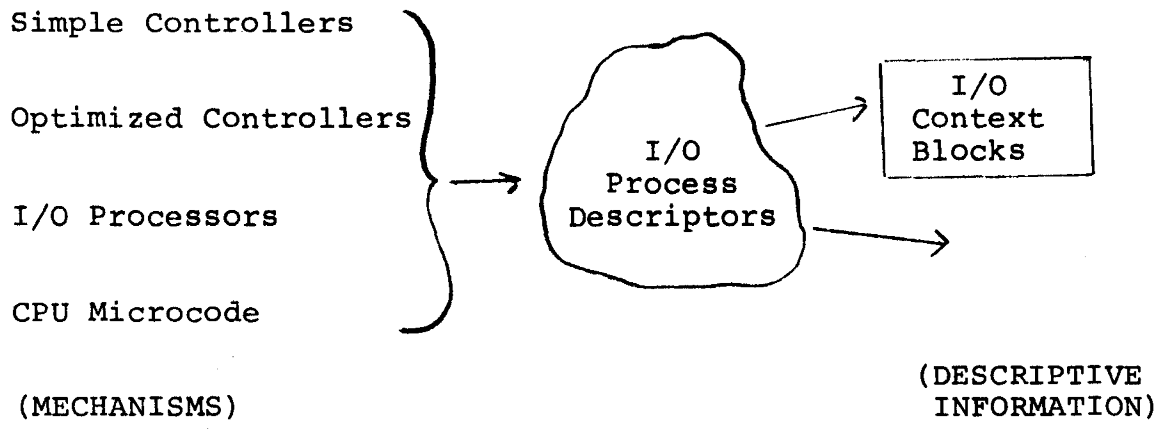
The operation of a PDP process, non-extended context, is binary compatible with existing machines. Furthermore, it is possible to mix extended and non-extended processes in the same environment (following the rules previously discussed).

The process structuring of I/O mechanisms is compatible with existing peripherals. There are no modifications required for CSR addresses, trap vector addresses, or UNIBUS characteristics. Furthermore, extended I/O devices can be added to and can co-exist with existing hardware.

The I/O aspects of the architecture are physically unrelated to the PDP-11 program aspects (although, logically, they directly relate insofar as they both have context which is accessed in similar ways, etc.). Consequently, advantage in the I/O area can be realized without the necessity of implementing microcode features in the CPU, and vice versa.

EXTENDABILITY

A major goal of the proposed architecture is the ability to extend I/O functions to allow (1) a higher performance optimized system for high-end machines, and (2) a cost-effective lower performance system for low-end machines. To do this, we make a clear distinction between (1) the descriptive information required to generically describe the I/O function, and (2) procedural information required to specify mechanisms and algorithms that perform the I/O function. Specifically, the architecture requires software to establish only the descriptive information, allowing wide variations in the actual I/O mechanisms that handle the I/O functions in accordance with varying performance requirements.



In this way, it becomes possible to extend the architecture to include wide variations of performance requirements on future PDP-11 systems.

To Gordon Bell

digital INTEROFFICE MEMORANDUM

TO: Distribution

DATE: January 22, 1975

FROM: Dave Nelson

DEPT: 11 Architecture/Planning

EXT: 4509 LOC: ML5/E67

COMPANY CONFIDENTIAL

SUBJ: PDP-11 ARCHITECTURAL ENHANCEMENT STRATEGY

This document is intended to focus on several architectural areas of PDP-11 systems which require improvements due to changes in technology, application, or competition.

The contents presented here generally falls into two categories:

1. Architectural modification/maintenance required for systems of current capability as a result of technology improvements, including:

   a. Faster context switching.
   b. WCS applications.
   c. ASCII console.
   d. Serial multidrop bus.

2. Architectural enhancements required to significantly improve system capabilities and performance, including:

   a. Virtual address space extension.
   b. Higher performance I/O system.
   c. Mapping hardware for virtual memory.
   d. Configurations for multiprocessing.

Considerable effort has been expended trying to categorize and compare the architecture of competitive machines, including the extent to which their operating software supports their architecture. Particular attention has been paid to MODCOMP, DG, HP, and INTERDATA.

At the same time, the structures of large scale machines have been analyzed in order to predict and evaluate architectural enhancements that will effect minicomputers in 3-5 years: in particular, MULTICS (virtual address space extension), IBM and Burroughs (high performance I/O systems, and virtual memory).

Emphasis has been placed on virtual address space extension and problems relating to the I/O system. Secondary areas include the interrupt system, writable control store, and multiprocessing.

That the 11 is becoming limited by its current virtual address space is a foregone conclusion. Several approaches to virtual address space extensions are included; however, recommendations which take into account compatibility, cost, and other architectural alternatives, are not yet fully developed.

Problems relating to the I/O architecture have not received as much attention, but it is likely that no single aspect of the PDP-11 is causing more problems: configurations of multiple controllers won't work (due to UNIBUS overruns); high-speed peripherals are being purposefully slowed down by sector interlacing; multiple controllers are being designed (UNIBUS and MASSbus) for the same peripherals; interfaces to MASSbus are expensive; overall performance is low; increasing data rates and configuration sizes are making things worse and worse.

Preliminary analysis of these problems leads one to conclude that: 1) The UNIBUS has adequate performance capability if intelligently controlled by an I/O processor; and 2) we need a serial bus for low-performance peripherals.

PDP-11 ARCHITECTURAL STRATEGY

SECTION I

I/O SYSTEM

I. PDP-11 I/O SYSTEM

Overview

The architecture of the PDP-11 I/O system has remained essentially unchanged since the inception of the PDP-11/20, over five years ago. Since that time, the PDP-11 has grown to support large operating systems in applications requiring high performance I/O throughput; and at the same time, the cost of low-end systems has dropped to the point where the UNIBUS is no longer cost-effective for many of its intended uses.

This section deals specifically to the following:

1. The PDP-11 has no data channeling capability which would allow gather write and scatter read operations. Consequently, operating systems are unable to load tasks to/from fragmented memory. Similarly, the lack of command chaining operations precludes the loading of programs to/from fragmented disk.

2. The UNIBUS, as currently used, has become obsolete: It is an overkill for most slow-speed devices (LP, CR, terminals); therefore, it is not cost-effective. In addition, it has inadequate bandwidth to effectively handle current mass storage devices (RP, RS, RK). The magnitude of this problem is best illustrated by our current practice of building high-speed peripheral devices, and then having to slow them down (by 2 or 3 sector interlacing) in order to use them on a UNIBUS.

3. While the UNIBUS appears to have inadequate bandwidth for our high-speed peripherals, the alternative of building multiple special high-bandwidth memory busses (à la 11/70) for I/O controllers does not appear to be cost-effective, considering the marginal performance gain that is realized in actual systems. Multiple high-speed controllers may eliminate data overruns, but they don't appreciably increase performance, and as such, price-performance competitive pressures will force us to abandon this brute force approach in favor of one which more efficiently utilizes the I/O facility.

4. The use of a high-speed serial bus for correcting unit record peripherals and terminals will afford significant advantages in interface costs, cable costs (over long distances), flexibility in configurations, great reliability, and better serviceability. A serial bus can be used as a systems bus in small, inexpensive applications (microprocessor, terminals, LP, floppy), providing minicomputer capability with some acceptable loss in performance.

Competition

I/O systems are difficult to compare and evaluate because of wide variations in concepts and objectives. The I/O architectures of H-P, MODCOMP, DG, INTERDATA, and IBM have been compared against features which are judged to be most important for high-performance, general-purpose, multiprogramming systems. Some of these features include:

	DG	H-P	MOD	INT	IBM	PDP-11	IOP-11
			IV	7/32			
fragmented memory I/O (data chaining)	Y	Y	Y	N	Y	N	Y
fragmented disk I/O (command chaining)	N	N	N	N	Y	N	Y
simultaneous block transfer	Y	Y	Y	N(1)	N(2)	Y(3)	Y(4)
channel optimization	N	N	N	N	Y	N	Y(5)

(1) Does have AUTO-DRIVER, implemented via CPU microcode, limited performance.

(2) No concurrency on some mpxr channel; can have multiple channels.

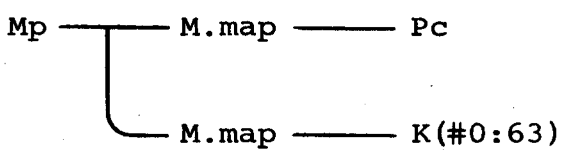
(3) Restricted by UNIBUS bandwidth, uncontrolled.

(4) MASSbus systems with wide band memory access only.

(5) Includes device optimization and request optimization, described herein.

THE FOLLOWING PAGES DESCRIBE THESE FEATURES
IN MORE DETAIL

Data General ECLIPSE



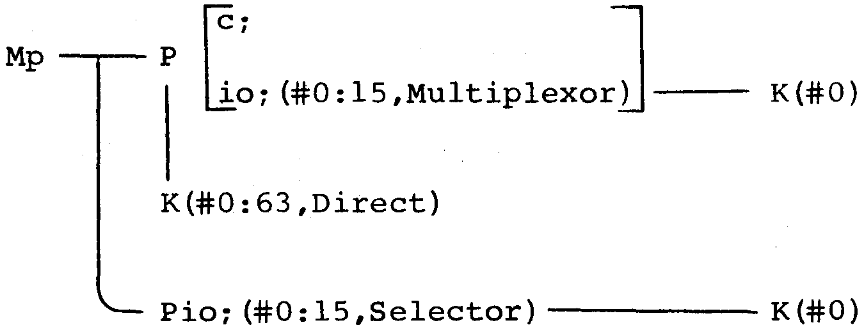
Features:

- o I/O is transferred to/from VIRTUAL mem space (allows fragmented memory).
- o Simple operation: Address and word count pair for each of up to 64 devices.

Disadvantages:

- o Although the MAP allows memory fragmentation, there is no command chaining that would allow disk fragmentation.
- o Like the PDP-11 I/O is asynchronous and uncoordinated. The sum of all device transfer rates is limited by peak bandwidth of the bus.
- o Any optimization is purely software controlled.
- o All I/O goes thru same map: restricted to 32K.

Hewlett Packard 3000



Features:

- o All I/O classified into 3 distinct categores.

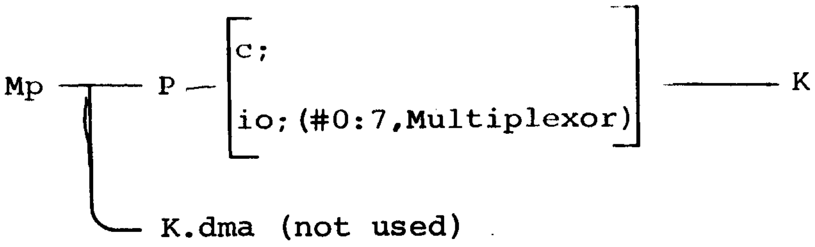
  - Direct for terminals, unit record, etc.
  - MPXR for intermediate speed disks, high speed unit received.
  - Selector for high speed mass storage.

- o MPXR and SELECTOR are programmable (ala IBM 360) and capable of command and data chaining.

Disadvantages:

- o Channel programs cannot cross device controllers.
- o Device controllers cannot connect to multiple channels (therefore, very little optimization is possible).
- o SELECTOR channel is busy for duration of a single channel program - little optimization possible.
- o No memory management support, except via chaining data.

MODCOMP IV



Features:

- Up to 16 subchannels, programmable with data chaining facility.

- Memory is paged with 256 word block size, so that constructing a channel program for a fragmented memory is straight forward.

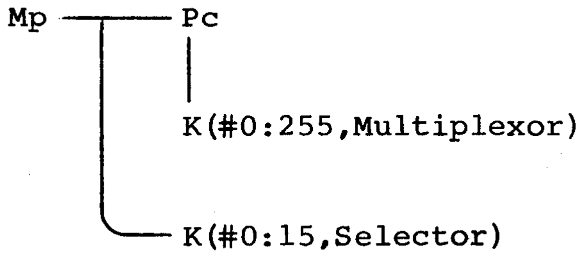
- DMA provided via separate memory ports.

Disadvantages:

- No command chaining facility.

- Only one channel program per device controller. Cannot easily optimize devices across multiple requests.

INTERDATA 7/32



Features:

- MPXR performs word and limited block transfers under program control.

- SELECTOR, controlled via MPXR, can perform high speed block transfers, directly to DMA port.

- "AUTO-DRIVER" channel converts interrupt devices into block transfer.

Disadvantages:

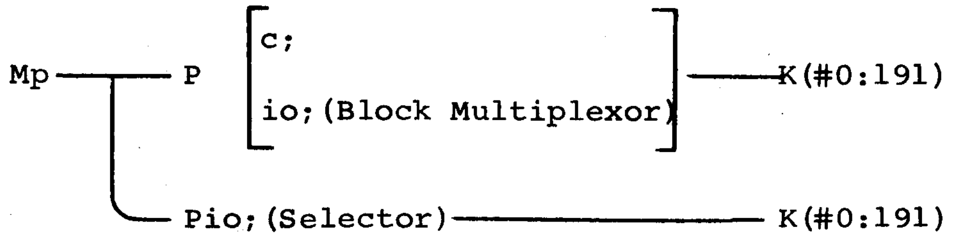
- No real channel capability for SELECTOR or MPXR devices; no chaining capability; no memory management.

- Sum of all device rates cannot exceed MPXR bandwidth (no burst mode capability).

- Only one device active on SELECTOR; channel busy for duration of request; channel optimization must be done in software.

- No large block transfer on MPXR, except using AUTO-DRIVER, which is low performance. Only one channel program per device.

IBM 370



Features:

- 64 subchannels operating under control of unique channel programs - complete chaining capability.

- Channel is busy only for duration of transfer. Arbitration is done at device level in real time, transparent to channel program. This optimizes channel utilization over 360 MPXR approach which arbitrated channel activity in software.

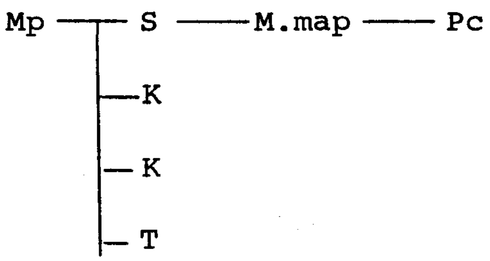
Disadvantages:

- I/O is done to/from physical memory requiring the exec to construct a channel program from memory maps.

- A controller/device can only be connected to a single channel program. Consequently, optimization of I/O requests from independent tasks is complex --- probably not feasible.

- Each device forces burst mode, utilizing 100% of channel regardless of its data rate.

Current PDP-11



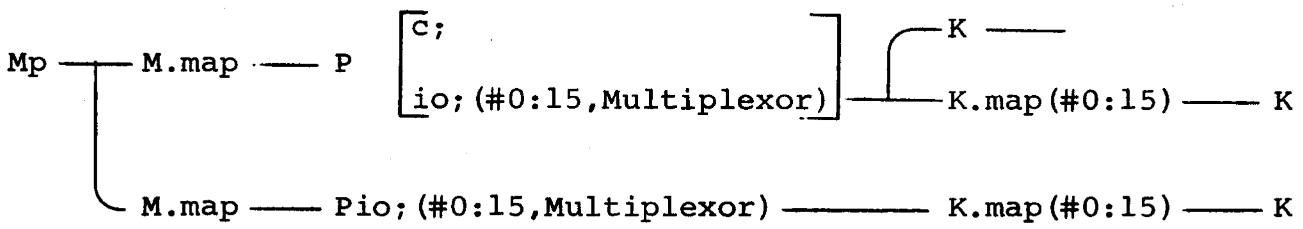
Features:

- o Simple operation: Address and word count pairs for each device.
- o All device types run on the same multiplexor bus.

Disadvantages:

- o No capability for block transfer to/from fragmented memory and fragmented mass storage devices.
- o I/O is asynchronous and uncoordinated: The sum of all device transfer rates is limited by the bandwidth of the UNIBUS.

Proposed PDP-11 with IOP-11



Mp — M.map — P [c; io;(#0:15,Multiplexor)] — K —

— K.map(#0:15) — K

M.map — Pio;(#0:15,Multiplexor) — K.map(#0:15) — K

Features:

- I/O transfers to/from virtual addresses. There can be an arbitrarily large number of maps, each for a different I/O process, no data chaining required.

- I/O transfers to/from virtual disk addresses by means of disk segmentation maps, eliminating need for command chaining.

- IOP controls positional and rotational optimization by multiplexing all I/O processes across all I/O devices. Channel is busy only for duration of transfer.

- Bandwidth permitting, IOP will schedule multiple MASS busses in accordance with device data rates to maximize bandwidth utilization.

- IOP will translate virtual block transfers into multiple physical block transfers and perform them in parallel.

- IOP will allow UNIBUS to perform word multiplexing operations and block multiplexing operations concurrently.

Summary of Conclusions

Two preliminary conclusions are derived from the studies to date: (1) It is possible to protect Digital's substantial investment in the UNIBUS over the next 3-5 years and immediately obtain a significant increase in system performance if we control the bus as a block multiplexor ,(by using an I/O processor with no hardware changes to bus or devices) and do away with sector interlacing, and (2) Design a serial bus as a more cost effective interface to low speed devices.

While an I/O processor will not substantially improve performance (only 50%), it will allow us to build large, high performance systems using the UNIBUS (unmodified bus and interfaces) and that the I/O systems on these configurations will perform better than machines built around high bandwidth memory/massbus connections (11/70). Further performance gain over present UNIBUS systems is realized by (1) Eliminating sector interlacing which will reduce transfer time for multiple sector transfers, and (2) Performing multiple physical requests for the same logical request in parallel.

I.1 I/O Busses

Figure I.1 shows the data rate of several PDP-11 busses (memory, massbuss, UNIBUS, etc., as a function of the cost of a typical interface. Superimposed on this graph is the data rates of several typical peripherals from which we conclude that the UNIBUS is a performance overkill (and not very cost effective) for line printers, card readers, and floppies. Also, the UNIBUS cannot adequately handle the data rates of high speed disks, requiring buffering and sector interlacing to effectively slow them down. Also plotted, is the data rate of the UNIBUS when controlled by an I/O processor, or when configured as a block multiplexor channel, which are described in sections I.2 and I.3.

I.2 I/O Processor Architecture

This section describes the architecture of a microprogrammed I/O processor capable of performing complete data channel operations between main memory and UNIBUS or MASSBUS mass storage peripherals.

The I/O processor is designed to multiplex a variable number of concurrent I/O requests in a manner which optimizes positional and rotational latency, and overlapped seeks.

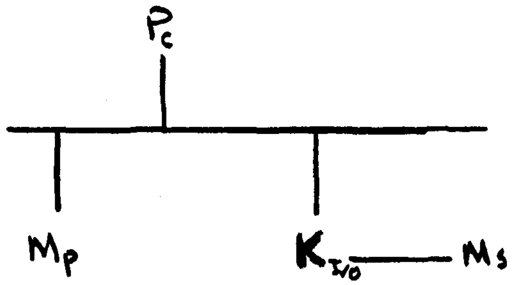
Each of the variable number of requests is described as independent transfers from independent virtual spaces; both main memory and disk memory are virtualized using memory maps having the structure of the KT-11 segmentation unit, and the FILES-11 retrieval pointer window, respectively. The disk map is a simple list of file segments and as such, it is adaptable to a variety of file structures.

Multiple I/O processors can be configured to perform multiple simultaneous I/O transfers. A single I/O processor could be microprogrammed to control multiple controllers, giving the functional equivalent of multiple I/O processors.

The concepts are shown to be extendable to unit-record and communication devices.

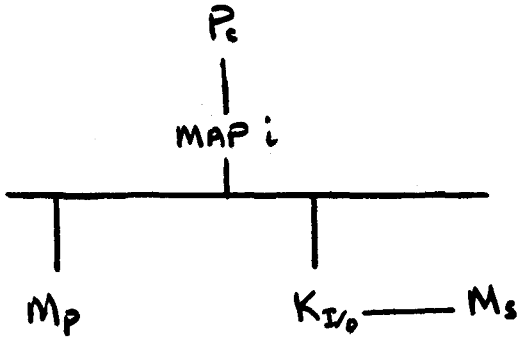
FORMAL DESCRIPTION

Original PDP-11/20 had a PMS* diagram like:

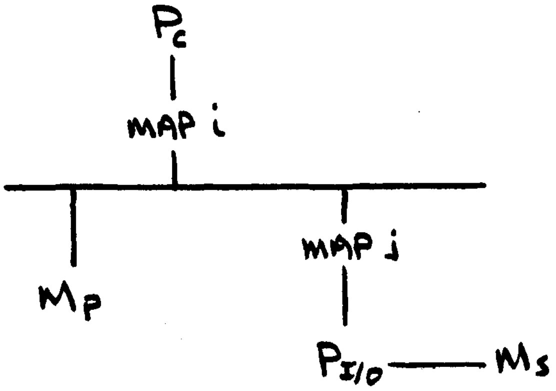


Here, all addresses between P_c & M_p, and between $P_{I/O}$, M_p & M_s are physical (no mapping). M_s = disk.

With KT-11 memory map option, we can consider each process, i, having its own set of map registers, so that with memory management we have

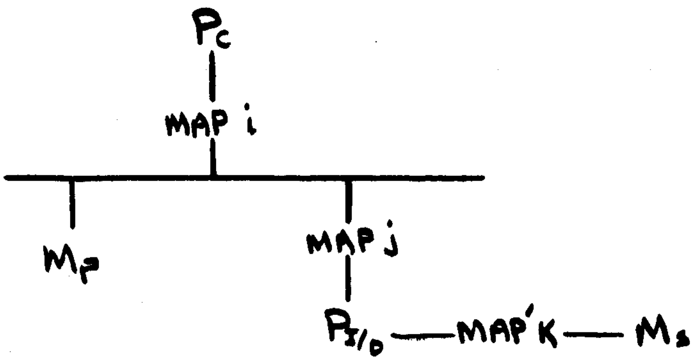


A first level enhancement over the above would be to allow a $P_{I/O}$ to operate through a MAPj so that memory management is seen symmetrically between P_c and $P_{I/O}$:



*Bell & Newell, Comp Str. P 15.

A second level improvement would allow $P_{I/O}$ to operate on M_s thru a MAP'$_k$, where MAP'$_k$ translates virtual block addresses into physical BLOCK addresses (analagous to the logic which currently exists in FILES-11):



This makes P_c symmetric to $P_{I/O}$, and M_p symmetric to M_s.

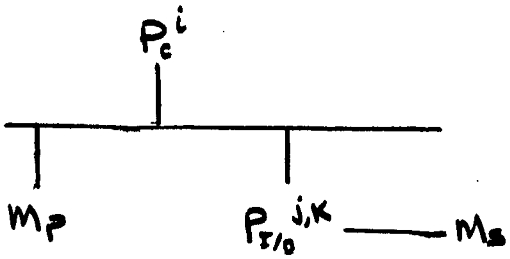
Now, if we let

$$\begin{bmatrix} P_c \\ | \\ \text{MAP } i \end{bmatrix} \equiv P_c^{i}$$

and

$$\begin{bmatrix} \text{MAP } j & \\ | & \\ P_{I/O} & \text{—— MAP}'K \end{bmatrix} \equiv P_{I/O}^{j,K}$$

then the above becomes:



which is same as original PDP-11, with virtual M_p, M_s.

What are the advantages?

- Advantages of $\left[P_c \setminus \mathrm{MAP}\ i\right]$ are well known to us. They allow us to address above 32K, and to map global, reentrant areas, etc, etc.

- Advantages of $\left[\mathrm{MAP}\ j \setminus P_{I/O}\right]$ would allow us to fragment physical memory (currently a restriction under RSX-11D) and transfer to M_s without processor (P_c) intervention. That is, it allows the equivalent of <u>data chaining</u>.

- Advantages of $\left[P_{I/O}\text{-}\mathrm{MAP}'K\right]$ would allow us to efficiently fragment disk (currently done in FILES-11, purely software). As above, we could transfer fragmented disk areas without processor intervention. This allows the equivalent of <u>command chaining</u>.

- Advantages of integrating both maps in $P_{I/O}$ allow direct transfer between memory and disk with arbitrary fragmentation. Advantages over traditional data channels (offered by MODCOMP and Interdata)are (1) maps are related to processes and files, and they already exist, (2) no data channel programming, and storage elements are symmetric.

The total logic involved in MAPj and MAP'_k is shown in figure 1.

Currently, the virtual M_p map exists exclusively in hardware (KT-11).

Also, currently the virtual M_s map exists exclusively in software (FILES-11).

What is proposed is a hardware/software trade off within each map scheme to optimize total performance.

The fundamental question relating to the implementation of the proposed architecture is: what portions of the organization should reside in hardware or software, P_c or $P_{I/O}$, and what are the interface relations between them?

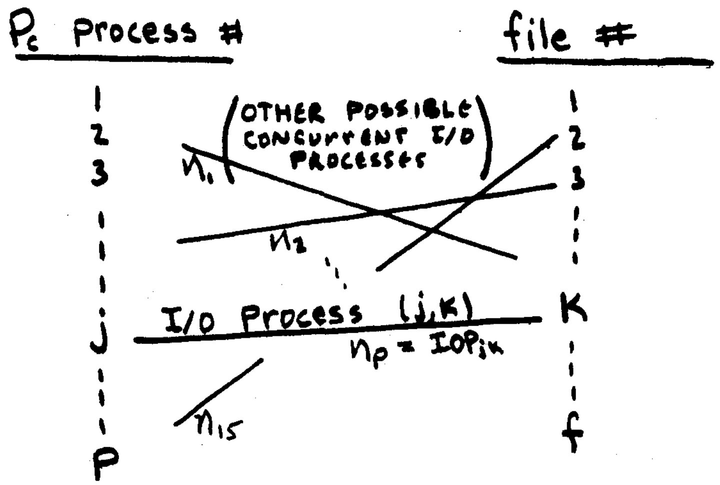
I/O ARCHITECTURE AND OPTIMIZATION

Formally, we have shown the fundamental I/O process to be represented by

$$M_P \text{ — } P_{I/O}^{j,K} \text{ — } M_s$$

where j represents a virtual memory transformation for process j, and k represents a virtual memory (disk) transformation for file k.

Together, the pair (j,k) describes one of several I/O processes which may be in operation at any given time. That is, in general, for P processes and F files, we can have:



Here, we have used the term I/O Process, IOP_{jk}, to denote a particular (j,k) pair representing I/O between two particular virtual spaces.

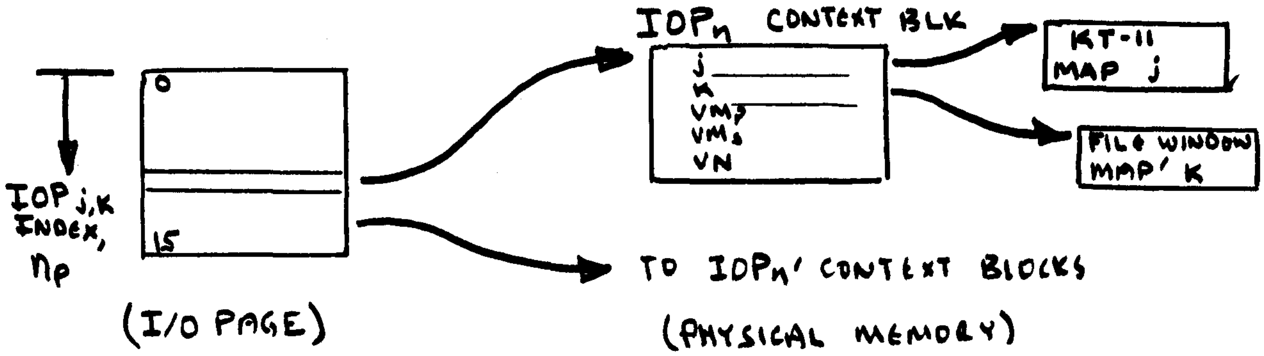
In general, then, we could have an arbitrary number of IOP_{jk} processes (not to be confused with P_c processes) at any given time. A particular device controller could be designed to handle a limited number of IOP_{jk}S, say up to 8. (The variability of this number allows a wide range of implementation [illegible] simple devices could be limited to 1; complex could have 16, etc).

The minimum context associated with an IOP_{jk} process is:

j	P_c PROCESS # (KT MAP PNTR)
K	FILE # (PNTR TO RETRIEVAL POINTERS)
VM_P	VIRTUAL MEMORY ADDRESS
VM_S	VIRTUAL FILE ADDRESS
VN	VIRTUAL WORD COUNT

The question remains as to what portion of this context should reside in memory, under management of P_c, and wnat should reside in the device controller, under management of $P_{I/O}$.

One possible hardware/software allocation of context is to have control blocks in memory (set up initially by software) whose physical address is loaded into a controller register.



The I/O processor (device controller) would then compute and maintain the following table:

DEVICE	CYL	TRK	SECTOR	MEM ADDRS	BLK COUNT	I/O

$IOP_{j,k}$ INDEX, n_p

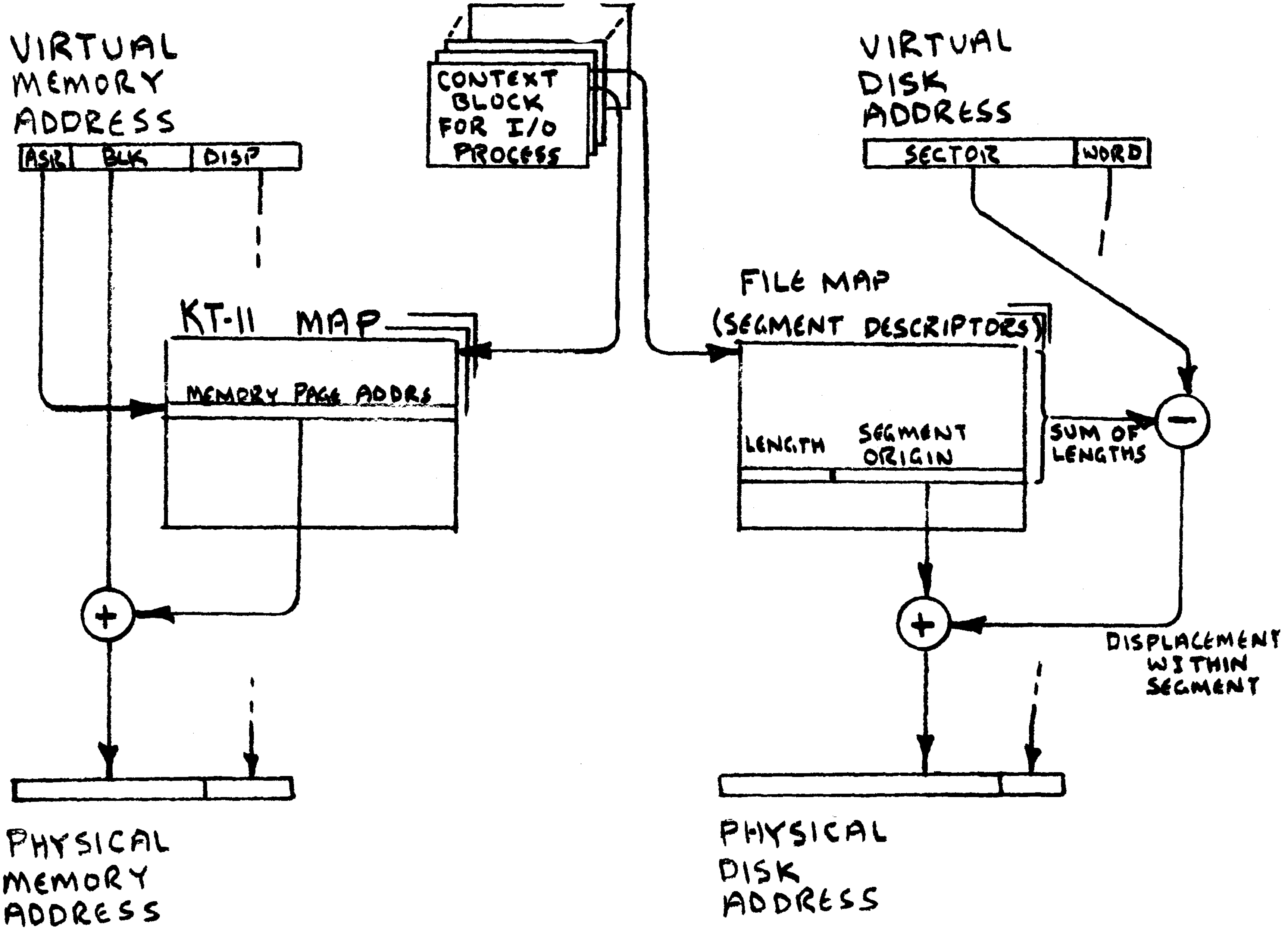


Fig I. 2 TRANSLATION OF {VIRTUAL TO PHYSICAL} ADDRESSES OF {PRIMARY SECONDARY} MEMORY

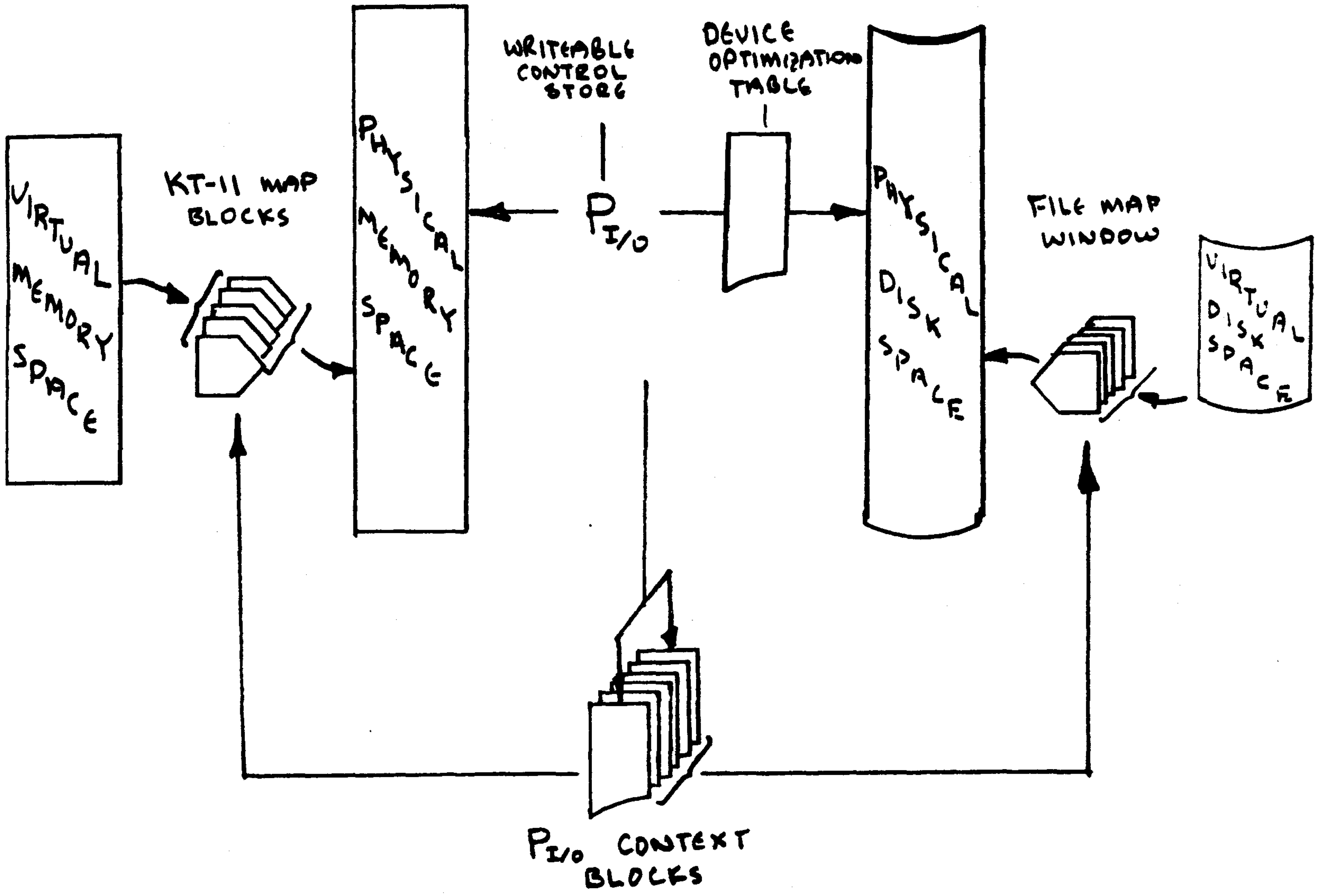


Fig 2. LOGICAL RELATIONSHIP OF {VIRTUAL / PHYSICAL} SPACE OF {PRIMARY / SECONDARY} MEMORY

POSITIONAL AND ROTATIONAL LATENCY OPTIMIZATION

We have described an architecture whereby the I/O processor is given a number of I/O requests to perform input or output between arbitrarily fragmented memory and disk. Each request is independent of others, and a typical I/O processor could be designed to handle up to 16 requests at any given time.

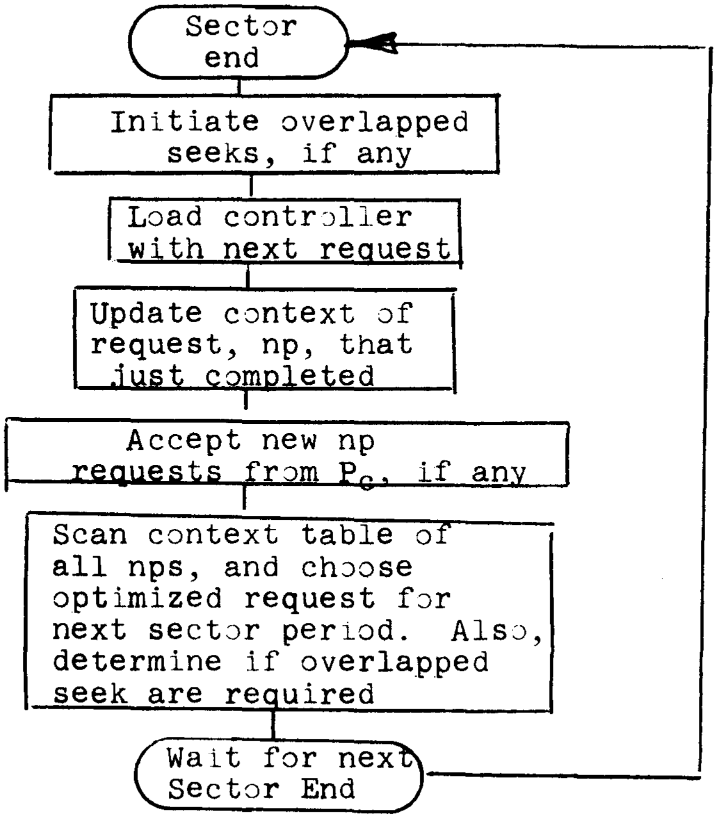
Now, since all requests are independent, and since their disk data is possibly fragmented, the I/O processor can be designed to multiplex all requests so as to optimize positional and rotational latency, and perform overlapped seeks for multi-device controllers.

We define an index, np, which identifies I/O requests

$$0 \leq np\ (i,j,vmp,vm_s,vn) \leq 15$$

where we have arbitrarily restricted the number of concurrent requests to 16.

For a multiple device, moving head disk controller, the I/O processor could be microprogrammed to perform the following sequences at the completion of each sector.


Sector end
Initiate overlapped seeks, if any
Load controller with next request
Update context of request, np, that just completed
Accept new np requests from P_c, if any
Scan context table of all nps, and choose optimized request for next sector period. Also, determine if overlapped seek are required
Wait for next Sector End

IMPLEMENTATION APPROACH

In general, I/O devices fall into one of the following categories:

(1) mass storage (random or sequential occurs)
(2) unit record (printers, readers, etc.)
(3) communication (terminals, synchronous lines, multidrop, etc.)

All devices in these categories could be implemented using similar I/O micro processors with writeable control store. Memory translation (virtual $M_p \longrightarrow$ Physical $M_p \longrightarrow$ Physical $M_s \longrightarrow$ Virtual M_s) has been divided into distinct context blocks, so that

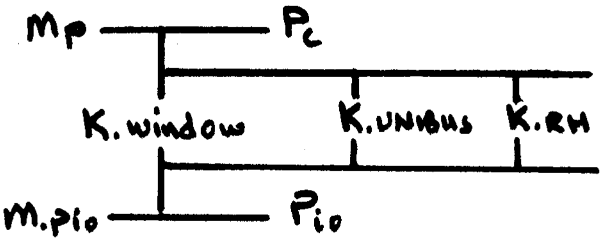
. mapping of primary memory is the same in all cases

. mapping of secondary memory space is device type dependent: for mass storage, we map file segments; for communications devices, we map onto lines; for unit record and magtapes, we don't map at all.

That is, one microprocessor approach with writeable control store could provide channel capability for most all device types.

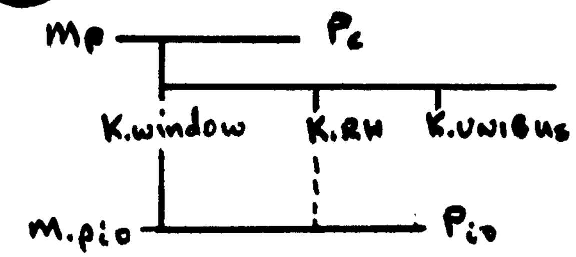
GENERIC FORMS OF PDP/IOP-11 STRUCTURES

(A)



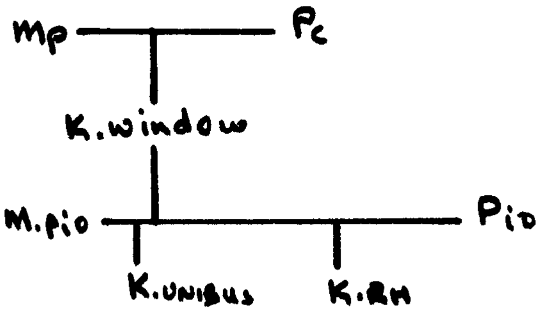
(data and control on separate busses)

(B)



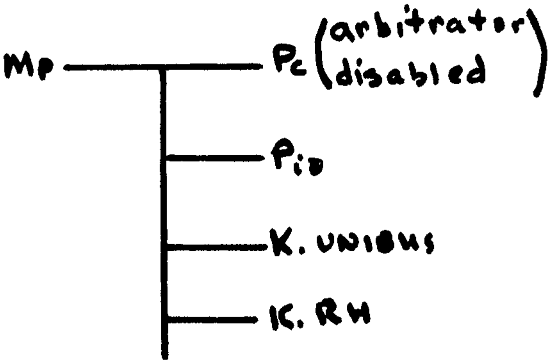
(data + control on central buss: interrupts xferred to P_{io} bus via window).

(C)



(data + control on P_{io} buss: data xferred to P_c bus via window)

(D)

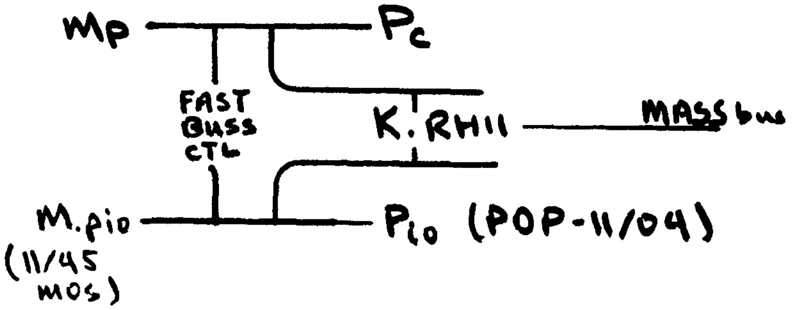


(data + control on P_c buss)

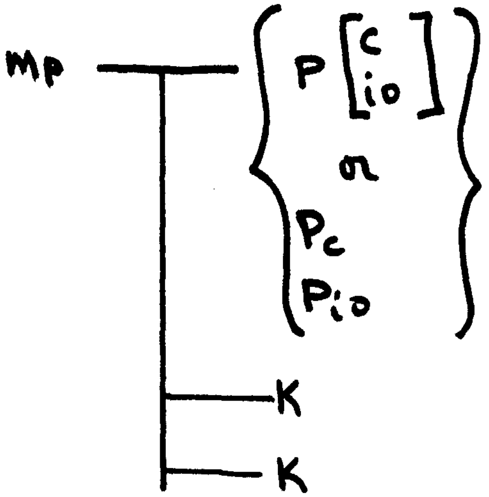
	A	B	C	D
HARDWARE ASPECTS				
o UNIBUS window required ?	Y	Y	Y	N
o dual port controllers required ?	Y	N	N	N
PERFORMANCE ASPECTS				
o does P_c-Mp bus get data directly ?	Y	N	N	Y
o does P_{io} bus get interrupts directly ?	Y	N	N	N
o does P_c & P_{io} compete for memory ?	N	N	N	Y
COMPATIBILITY ASPECTS				
o compalible without reconfiguration ?	N	Y	Y	N

SPECIFIC STRUCTURES BASED ON EXISTING CPUS.

PDP-11/45 (generic form A)

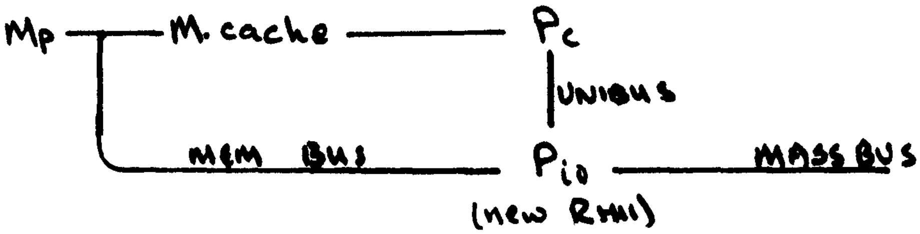


PDP-11/PDQ (generic form D)



(using writeable control store, or an additional processor)

PDP-11/70 (generic form A)



I.3 Performance of Block Multiplexors

Section I.3 is a preliminary design of an I/O processor designed to optimize data channeling activity on medium scale 11 systems. This section compares the theoretical data used by IBM to design their Block Multiplexor Channel (IBM Systems Journal, Vol. 11, #3, 1972), with a theoretical study of DECsystem-10 performance (Turner, Stone, Disk Throughput Estimation, 1071), and actual measurements on CS-2 (Turner).

The IBM data agrees well with ours (less than 50% variation in total throughput as a function of Q lengths, number of devices, channel architecture, etc.), and sufficient confidence has been attained in their applicability to PDP-11 systems to draw the following preliminary conclusions:

1. In terms of performance, IBM's Block Multiplexor Channel is equivalent to an I/O processor, although the architectual approaches are quite different.

2. I/O processors don't help matters as much as we tend to think they would. On PDP-11 scale systems, we might increase throughput by 50%. However, for certain applications, and systems (demand paging), throughput could increase 100%.

3. But, an I/O processor is considerably better than multiple MASSBUS controllers, even when the MASSBUS is programmed for overlapped seeks. IBM data show that a single Block Multiplexor Channel (performing one I/O operation at a time) can outperform eight MASSBUS type controllers, on a system with 64 devices.

4. The UNIBUS, under control of an I/O processor, can run similarly to IBM's Block Multiplexor Channel at a rate greater than a million words per second which is more than any of our mass storage devices. Consequently, an arbitrarily large configuration of RS04's, RP04's, etc., all connected to the UNIBUS (unmidified) can out perform the same configuration connected by up to eight MASSBUS controllers having wideband (non UNIBUS) access to memory.

An I/O processor (separate micro controller, or integrated into CPU microcode) is a more cost effective way of managing mass storage devices than are multiple MASSBUS's controlled by software. The UNIBUS, on an 11/40 size machine, has sufficient horsepower to out perform the largest configuration (4 MASSBUSSES)of the PDP-11/70.

Figure 4 Simulation results using a variable number of IBM 3330-like devices on a single channel

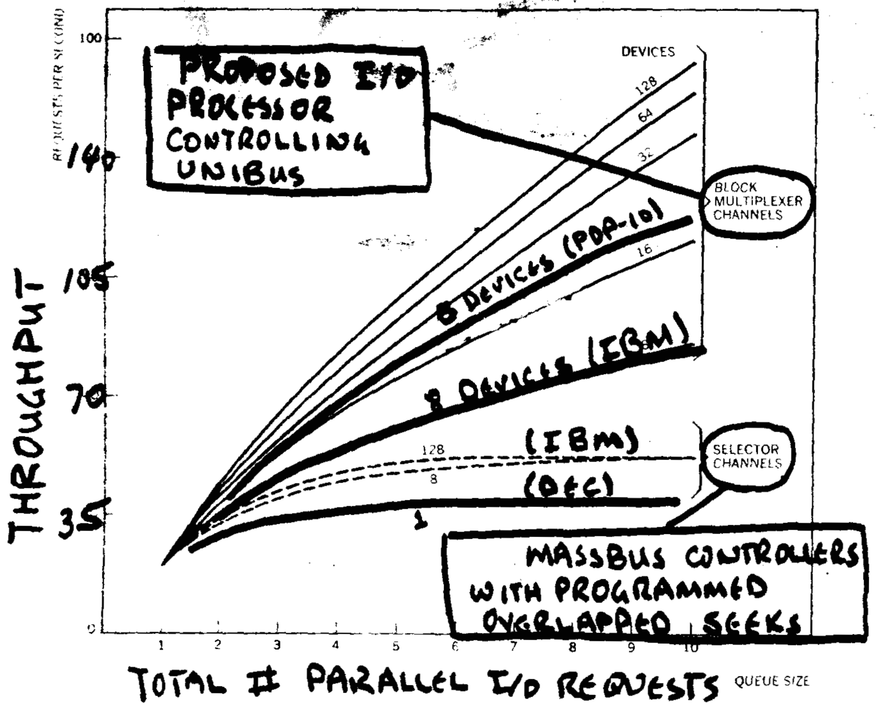
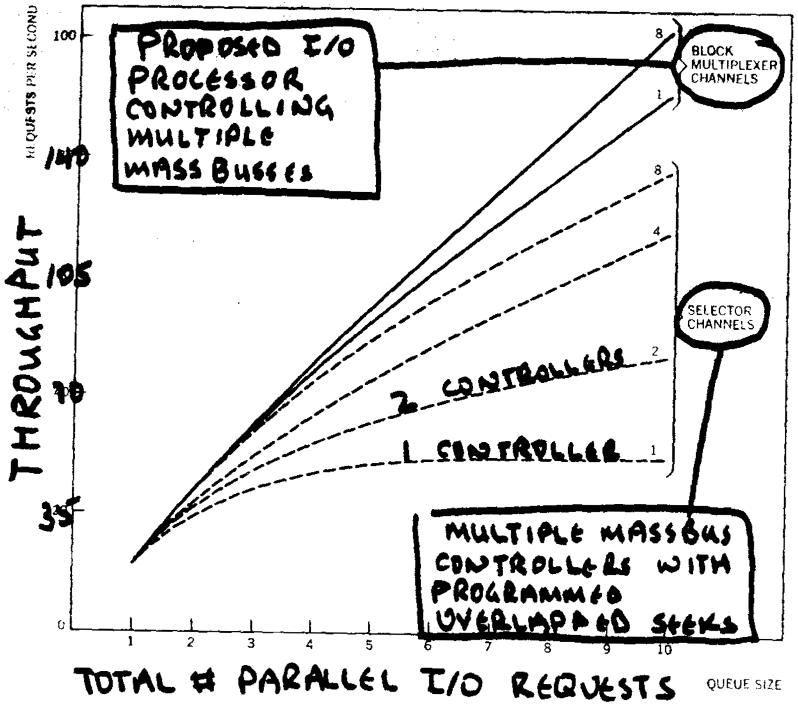


Figure 5 Simulation results using IBM 3330-like devices evenly distributed among a variable number of channels

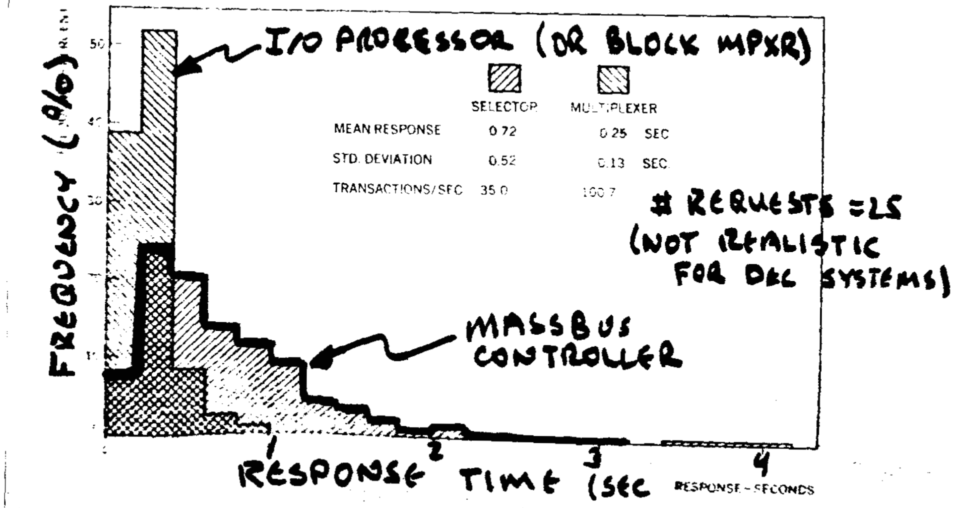


SIMULATION DATA OF PDP-10.

# REQUESTS	# DEVICES 1	2	4	8
1	20			
2	25	40		
4	30	45	66	
8	35	50	74	106
16	40	55	80	116

THROUGHPUT (BLKS/SEC)

Figure 6 I/O response time distributions for a constant queue of 25



	SELECTOR	MULTIPLEXER	
MEAN RESPONSE	0.72	0.25	SEC
STD. DEVIATION	0.52	0.13	SEC
TRANSACTIONS/SEC	35.0	100.7	

COMPARISON OF IBM/DEC DATA SHOWING THROUGHPUT ADVANTAGES OF I/O PROCESSORS AND BLOCK MUXS.

I.4 Simulation of I/O System

Current efforts include a digital simulation of the PDP-11 I/O system. These results will measure:

1. Performance gain for I/O processor and multiple controller configurations.

2. Effects of mixing device types on the same MASSBUS controller.

3. Throughput as a function of request que length for for typical configurations.

4. Configurations and activities that characterize data late conditions.

This section will be included when specific results are obtained.

I.5 Serial Bus

The serial multidrop bus is intended to provide:

1. A low cost systems buss for small processors at an acceptable performance level.

2. A cost effective means of connecting multiprocessor systems.

3. A means of connecting large numbers of terminals.

4. A cost effective bus for unit record peripherals for medium and large scale 11's.

Figure I.5.1 shows the organization of the bus arbitrator in relation to each node. The arbitrator is a processor with a buffer memory used for communication among nodes. Transfers to and from node zero uniquely bypass the buffer memory, and as such, node zero may be used to connect the central processor in a hierarchial system.

Figure I.5.2 shows the variety of interconnections possible with the serial bus protocol. Multidrop nodes can connect to remote multidrop nodes, a low cost dedicated interprocessor link (IPL), or to microprocessor based computer systems.

DISTRIBUTED SERIAL BUS CONFIGURATION

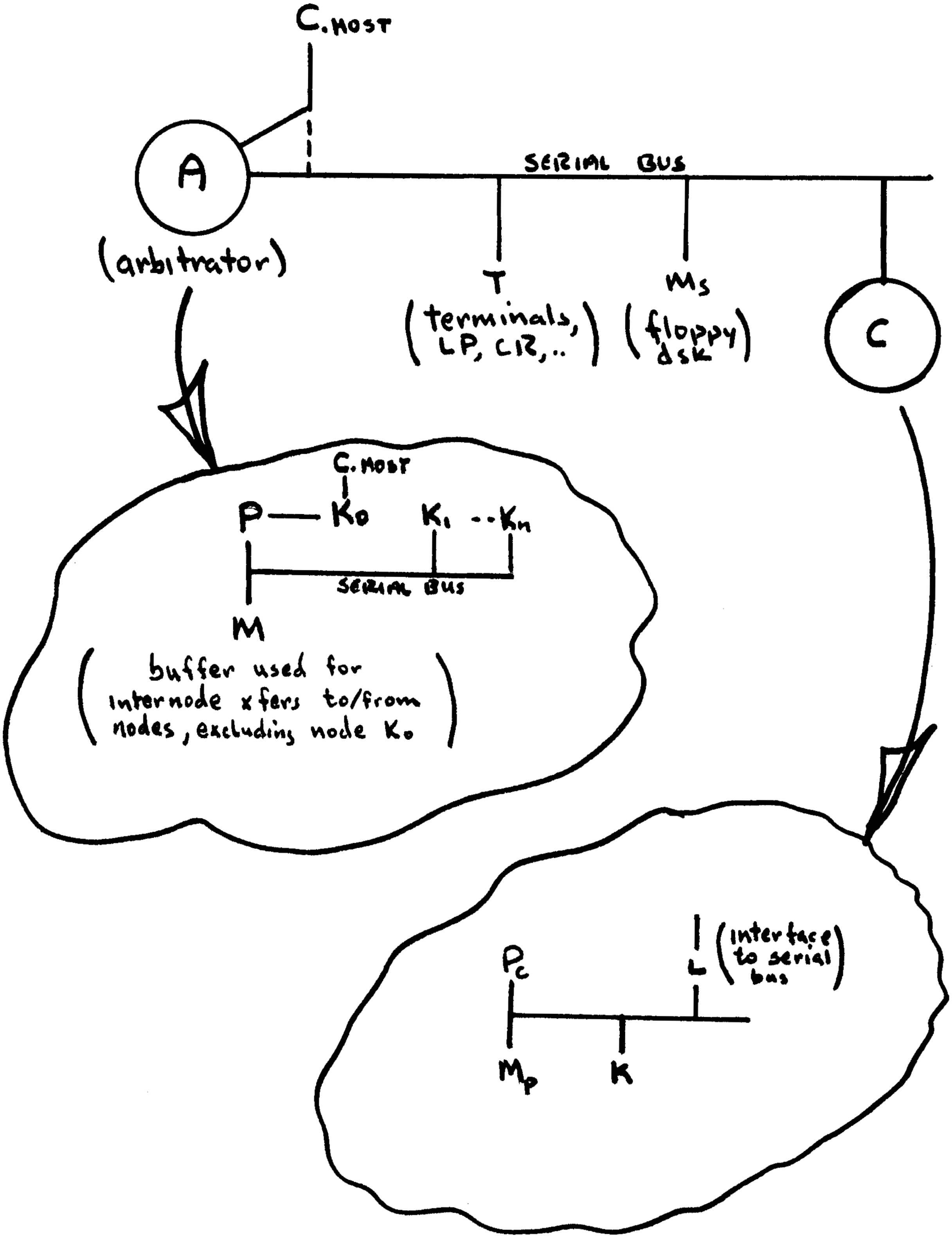


figure I.5.1

POSSIBLE MULTIPROCESSOR LINKS

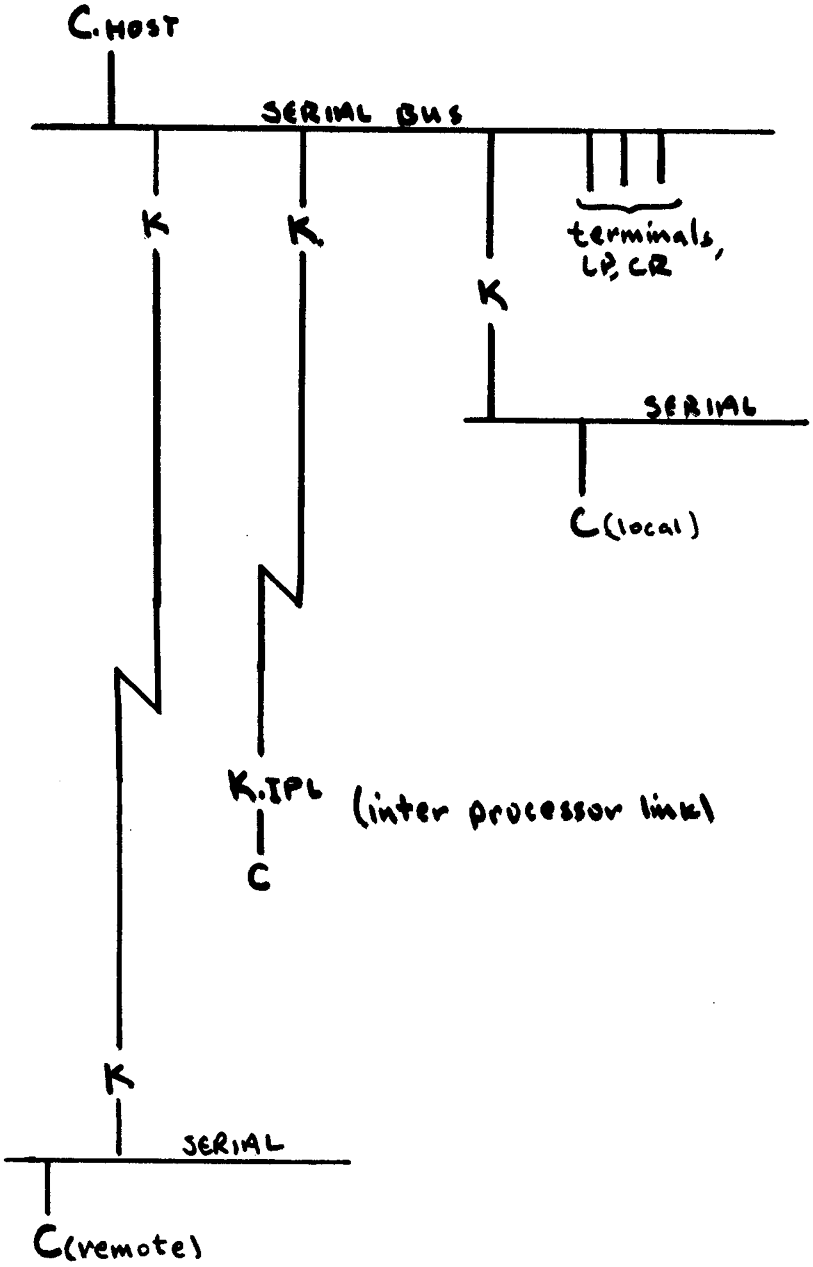


figure I.5.2

SECTION II

INTERRUPT SYSTEM

II. INTERRUPT SYSTEM

Overview

The use of a stack for saving and restoring machine registers in the PDP-11 interrupt system works well for small single user un-mapped operating systems. In large mapped memory systems, running multiple processes, however, some improvements could be made to reduce the time overhead of saving and restoring machine context.

Presented here is an architectural description which contains many desired attributes for such systems, while retaining a significant level of compatability with existing machines.

Competition and Motivation

The motivations for this section include the following:

1. Both Modcomp and Data General have announced machines which incorporate interrupt features which are of superior performance to the PDP-11. These features include switching of register blocks and memory maps in hardware.

2. The existing KT-11 architecture is inadequate for (a) Larger virtual addresses, (b) More efficient memory allocation, (c) High performance demand paging. Improved memory management architecture will require different techniques for changing the memory mapping context. These changes explicitely affect interrupt processing.

3. Operating Systems which are process structured (RSX-11D) impose a high degree of context switching among processes with significant overhead. Context switching and interrupt servicing are highly related and require an integrated approach.

4. The PDP-11 architecture does not lend itself toward high speed communications and interrupt processing. Interrupt-by-character devices run with excessive overhead, and while the attached proposal does not directly deal with this problem, it does provide the groundwork. Further to this point is the growing requirement for hardware managed I/O functions such as command and data chaining.

II.1 Formal Description

Formally, a computer system operating in complex environment can best be described in terms of "processes" (rather than programs and variables, etc.). A process is defined as a "procedure with context" where procedure denotes the machine time-sequence structure, and context denotes the processes state. Context infers different things in different environments, but in terms of the hardware processor, context generally refers to the general registers, map registers, and Program Status Word.

Emerging from the concepts of processes is the process number. Symmetry behooves us to treat all processes alike (except for priority), and therefore, since a process can be anything from a FORTRAN program to an interrupt service routine, the process number serves to distinguish them from within the system.

In the simplest sense, process numbers should be used as an index into a table of context vectors. Each vector contains a pointer to a context block which holds the machine state for the process.

Interrupts in such a system are simply effected by asserting a new process number on the CPU, which, upon arbitrating the priority of the requested and running processes, may choose to invoke a context switch to the new requested process.

Now, specific to the proposed PDP-11 interrupt system, the process number, P, is used as an index into the CONTEXT VECTOR TABLE (origined at physical location zero) to obtain the CONTEXT VECTOR, CVP. Currently, these locations are the trap vectors which contain the new PC, PS. Under the new definition, however, these locations would contain the physical address of the CONTEXT BLOCK (a mode bit in the PS could control this redefinition and thereby retain full compatibility).

Referring to figure II.2, MAP contains the location of the memory map registers (KT-11 registers or equivalent). SP and PC contain virtual addresses of the processes stack pointer and program counter, etc.

A new register located in the I/O page would be assigned to hold the CURRENT CONTEXT VECTOR, CCV. Upon interrupt, the processor would save the existing context in the current processes CONTEXT BLOCK, load the context addressed by the CONTEXT VECTOR, and put the old CCV on the new processes stack. A return from interrupt (RTI or RTT), or a fake (software generated) RTI, would pop the old CCV off the stack and restore the context of the old process.

II.2 Interrupt:

a. Store context in block addressed by CCV. Save CCV as CCVOLD.

b. Load CCV from CONTEXT VECTOR TABLE indexed by process number (currently the trap address).

c. Load machine context from block addressed by CCV.

d. Push CCVOLD onto new stack.

II.3 Return from Interrupt:

a. POP CCVOLD from current stack.

b. Store machine context into block addressed by CCV.

c. Load CCV with CCVOLD.

d. Load machine context from block addressed by CCV.

II.4 Compatibility and Extendibility:

Compatibility with existing peripherals is retained.

Compatibility with existing software is controlled by a single mode bit. Modifications to existing software to take advantage of improvements are minor.

Implementation logic is similar in that both modes involve retrieval of vectors, pushing and popping variables onto stacks, etc. Block or cache loading is, of course, new.

The concept naturally extends to scheduling of software tasks, thereby reducing the time/core overhead of loading/restoring machine registers for every context switch. Admittedly, this is a small portion of a tasks context, but would serve to speed up switching significantly.

The architecture allows for high performance implementations which would "cache" the machines registers from context blocks, thereby saving and restoring only those registers which were used by a process.

The architecture allows for low performance implementations which would treat the machine registers as ordinary memory, thereby reducing the number of internal CPU registers.

The architecture allows for a simplified implementation whereby all memory and machine registers are accessed through a common cache. Access frequencies of the general registers would naturally retain them in the fast access locations, and performance could be retained by disabling memory updating on all register addressing modes.

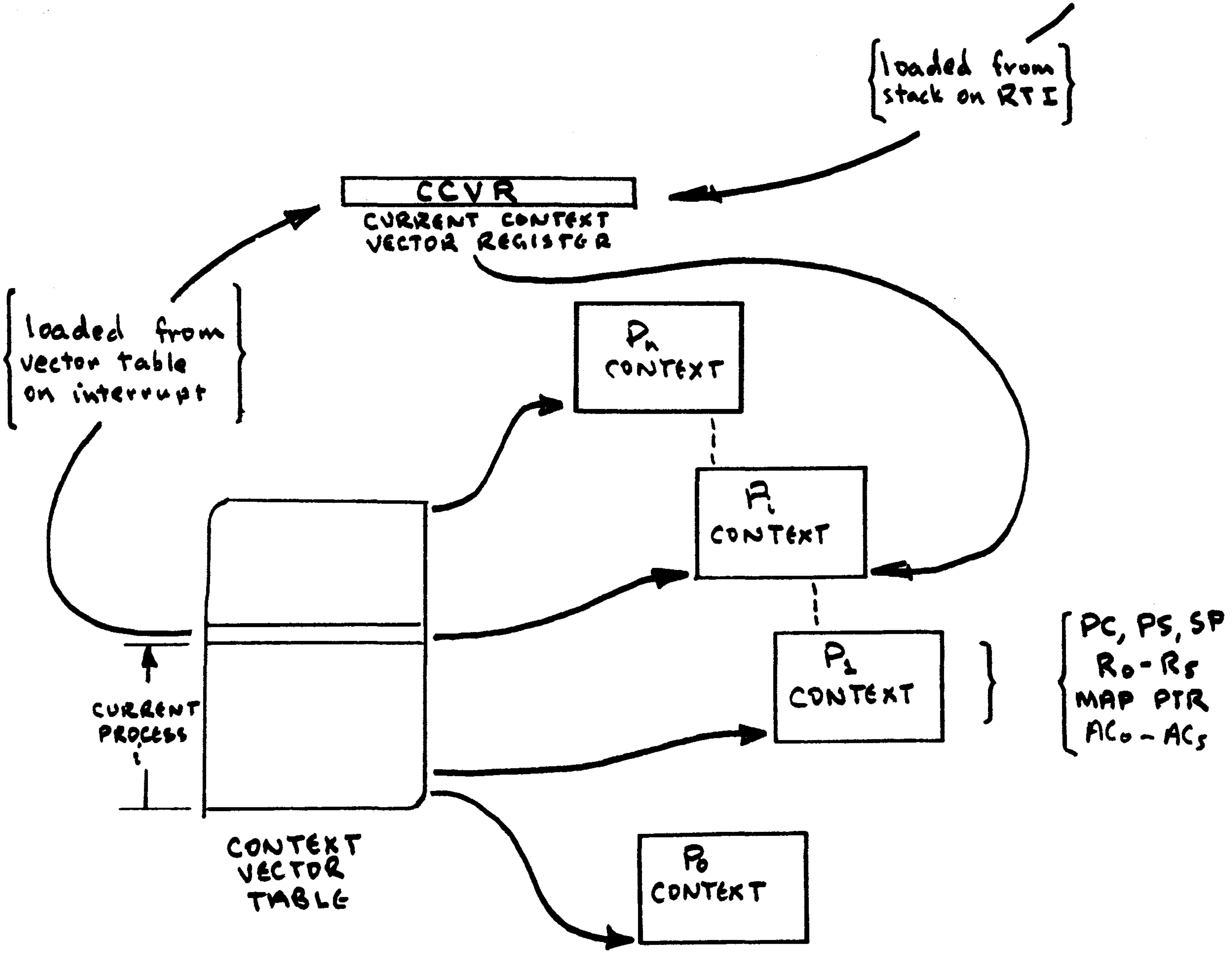
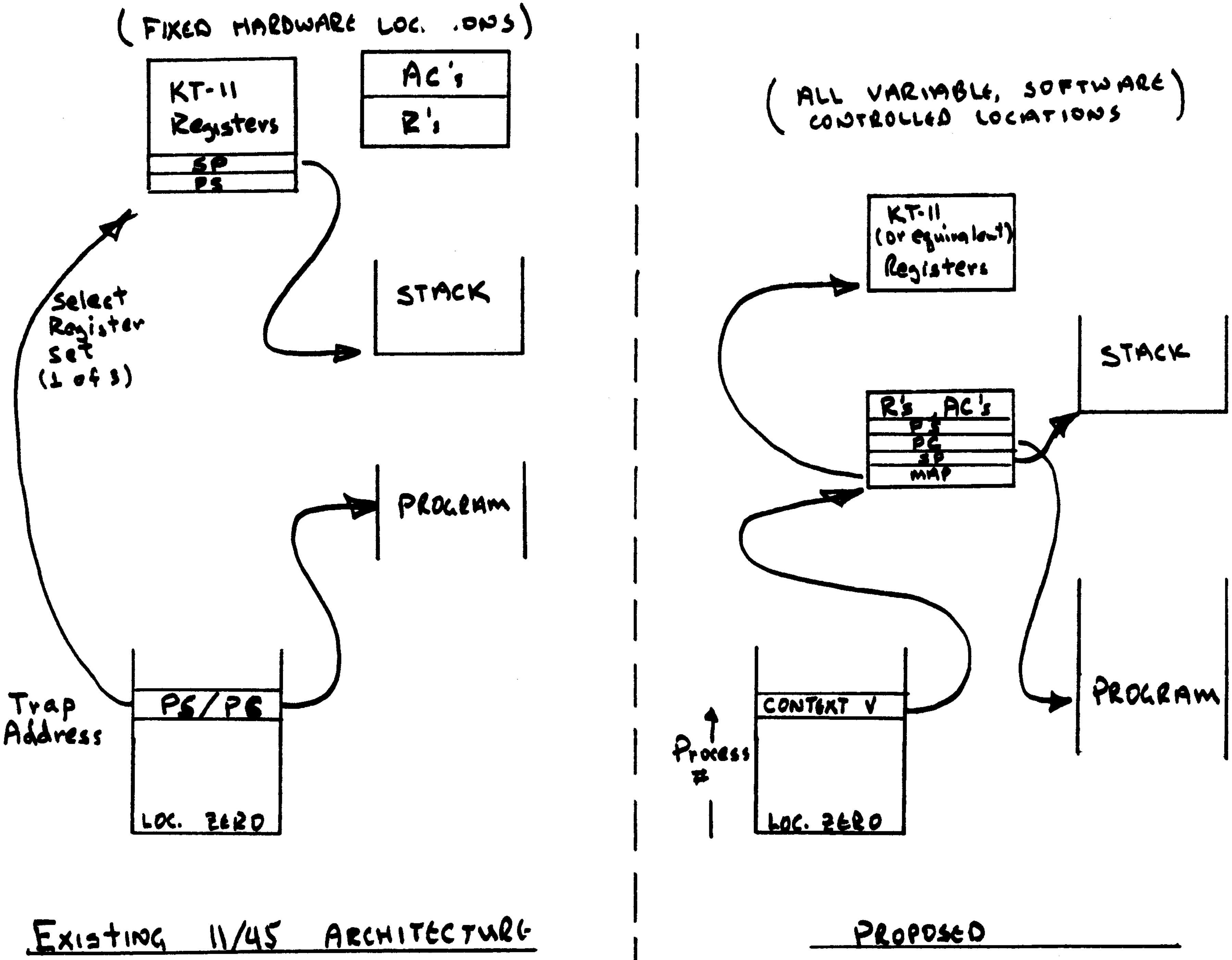

loaded from stack on RTI
CCVR
CURRENT CONTEXT VECTOR REGISTER
loaded from vector table on interrupt
P_n CONTEXT
P_i CONTEXT
P_1 CONTEXT
P_0 CONTEXT
PC, PS, SP
$R_0 - R_5$
MAP PTR
$AC_0 - AC_5$
CURRENT PROCESS i
CONTEXT VECTOR TABLE

Figure II.1



Existing 11/45 Architecture

Proposed

Figure II.2. A Comparison

Appendix A

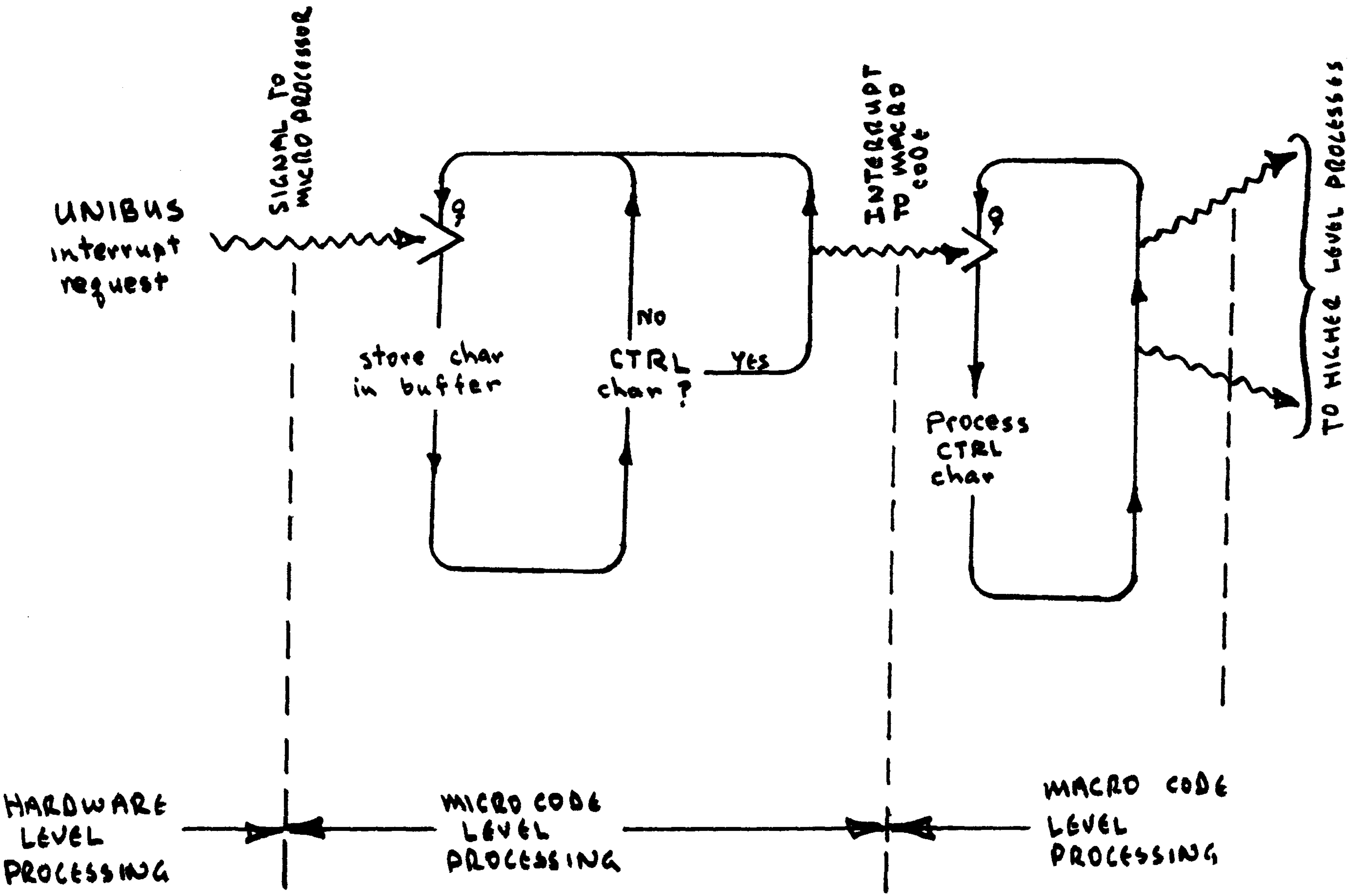
The PDP-11 is in need of some hardware enhancements in the I/O area to facilitate command and data channing (like data channels), and high speed communications channels. But, we should stop thinking of data channels as some kind of unique processors. Rather, we should consider the general I/O function as simply another process within the system. Currently these I/O processes are our interrupt service routines, but they are too slow for much of the I/O because they run at macro level. A better approach is to run the high throughput, short, interrupt routine processes at micro level with the device context stored at the macro level. Advantages of this approach include higher throughput and the ability to build transfer-by-word devices and program them as NPR devices. We have a lot to learn in this area, but in many subtle ways, the idea of processes having their machine state contained in memory and addressable by a process number provides a structural framework that relates well to the general I/O problem.

The following description of an asynchrounous line controller is given by way of example and is not intended as a serious proposal. The communication device described is an asynchrounous line multiplexor that generates a BR request for each character. In order to minimize the overhead of processing each character, it is desired to implement a micro level controller that buffers all characters in separate line buffers, and generates an interrupt when any CTRL character is encountered.

The software handler is designed to set up buffers and process control characters. In doing so, it loads the address of a table of buffer pointers into R_O and then executes an RTI. The microprocessor would be programed to store each character recieved into the appropriate line buffer. This is accomplished in microcode by implicitly executing the equivalent of :

```
MOV        line (RO), Rl        ;get address of buffer
MOV        ch, @(Rl) +          ;store in buffer
MOV        Rl, line (Ro)        ;update address
BIT        ch, CTRL             ;test character
BEQ        exit                 ;exit
(generate macro interrupt)
```

Since the above instructions are processed at micro level without having to undergo a macro level context switch, the character processing rate is comparable to an NPR controller. An additional advantage is the ability to generate macro level interrupts for control characters, where higher level processing is required.



PROCESS DIAGRAM FOR MICRO CONTROLLED COMMUNICATIONS CONTROLLER.

SECTION III

VIRTUAL MEMORY SYSTEMS

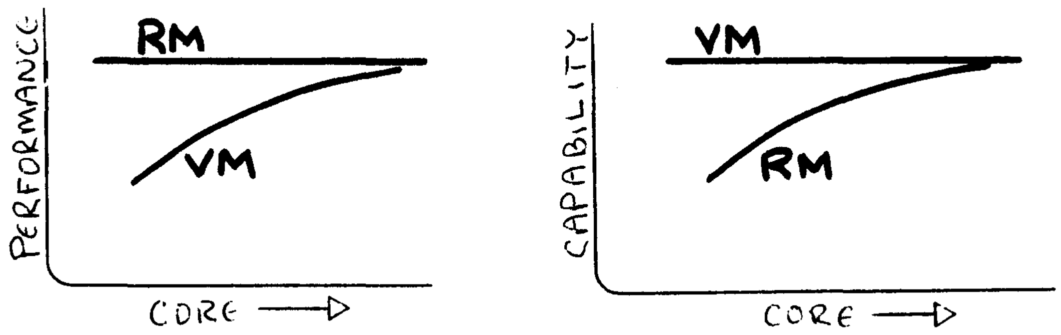
III Virtual Memory

The idea of using a backing store (disk) to provide a virtual memory (VM) environment has been around for about 15 years. Since the late 1960's, more than 200 research papers have been published on the subject, and of the 50 papers published in IBM's Systems Journal since 1971, no less than 10 (20%) have been devoted to VM. The advantages and disadvantages of VM are becoming well understood, and there are growing indications that mini-computer manufacturers (Modcomp, Data General) will be implementing VM in the near future, as evidenced by their recently announced memory management architectures.

The purpose of this section is twofold: first, to focus attention on the potential that VM offers not only as RSX-11D size systems, but also for RSX-11M and RT-11 size systems, and second, to expose the inadequacies of our current KT11 memory management hardware for use in VM systems. Contained herein is a proposed memory management unit which is claimed to be better suited for demand paging as well as physical and virtual memory allocation. It is also cheaper to build, faster in performance, and adaptable to both small 16-bit machines as well as large 32 bit machines.

III. 1 Some Characteristics of VM Systems

Virtual memory (VM) systems can be characterized with respect to real memory (RM) systems by distinguishing levels of capability from levels of performance. In this sense, we consider "capability" to be the types of functions that the system can perform without regard to the rate at which it does it. By the same token, we consider the machines "performance" to be the raw speed of the machine without regard to its capability. Given this, we can compare the relative levels of capability and performance that can be attained by VM and RM machines as a function of the system's real memory size.

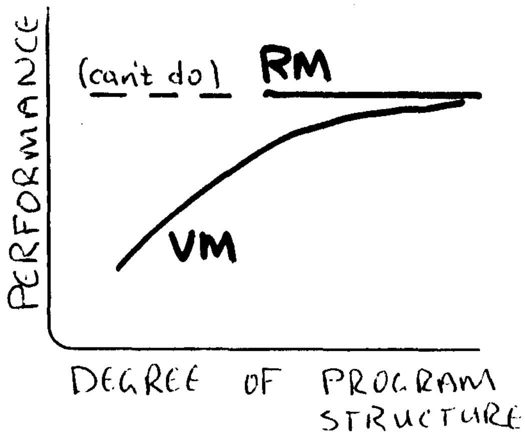


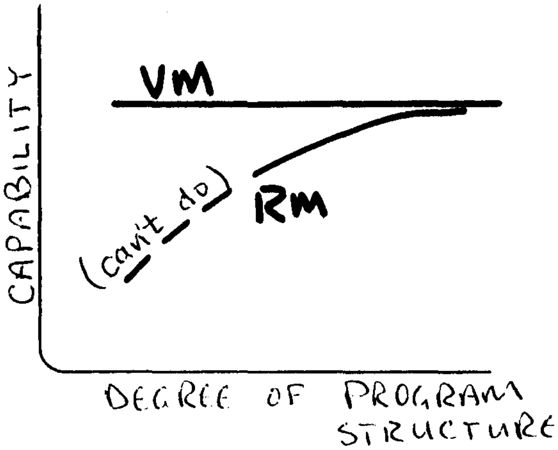
This simplified comparison shows that whereas the capability of an RM system varies significantly as a function of core (while its performance is constant), the capability of a VM system is constant as a function of core (while its performance varies). The key point here is that it is easier and more cost-effective to build systems whose capabilities are independent of the amount of core. The marketing and engineering departments of Digital have become overly preoccupied with the significance of the amount of core memory in its relation to functional characteristics. Rather than treating real memory as just another level in an integrated memory hierarchy, we continue to perceive it as a hard resource/facility which is critically related to functional capability.

III. 2 Advantages of VM to Digital

It is tempting to suggest that VM offers the advantage of reducing software development costs, since programs could be written without regard to physical memory size. While this is true to some extent, it is important to note that a considerable portion of the research done in VM systems has been concerned with effects of program structure and program structuring techniques aimed at offsetting the performance degradation inherent in VM systems. It has become clear that virtual memory is far from "transparent", and that in order to recover the performance loss, the software engineering effort required to appropriately structure programs will likely be comparable to the effort we currently expend in coping with real memory overlaying techniques.

The primary advantage that VM gives us is the option to structure programs rather than the necessity to do so. This option allows us to write large, highly capable programs which are either (a) unstructured with low performance, or (b) highly structured for high performance or some intermediate level depending on the intended frequency of use. Thus, under VM, one applies program structuring techniques in order to achieve a performance level, rather than a capability level. On the other hand, under RM, large program capability, and therefore performance, is impossible without some degree of program structuring (overlaying).





III. 3 Memory Management Unit

There have been numerous research studies published regarding the optimization of various parameters for virtual memory performance. Generally speaking, the results of these studies show that the performance of VM systems is:

(a) Very sensitive to program structure characteristics
(b) Moderately sensitive to page size
(c) Slightly sensitive to page replacement technique

Performance is generally optimized when the page size is comparable to the characteristic program structure size. Unstructured programs favor a smaller page size, whereas highly structured programs favor a larger page size. IBM studies have shown that as long as the program is structured in accordance with the page size, then larger page sizes are disproportionately preferred (Hatfield, Journal of R/D).

While there are variations in optimized page size values, there is general agreement that the KT-11/D page frame (8K bytes) is too large, and that a more appropriate value is 1K-2K bytes. Figure III. 1 compares the address translation logic of the KT-11/D and a proposed KT-11/X. It should be noted that the smaller 1K page size of the KT-11/X is sufficiently small so that physical memory can also be allocated in 1K sizes. This ability to allocate physical space in the same size as virtual space not only simplifies system software, but also eliminates the complexity and latency required for the extra addition that's currently required the KT-11/D.

Conclusion

Virtual memory on PDP-11 systems should be considered as part of a 2-3 year systems strategy. While changes in memory and backing store technologies may change some parameters of VM, the basic concepts and economies of memory hierarchies will long be with us.

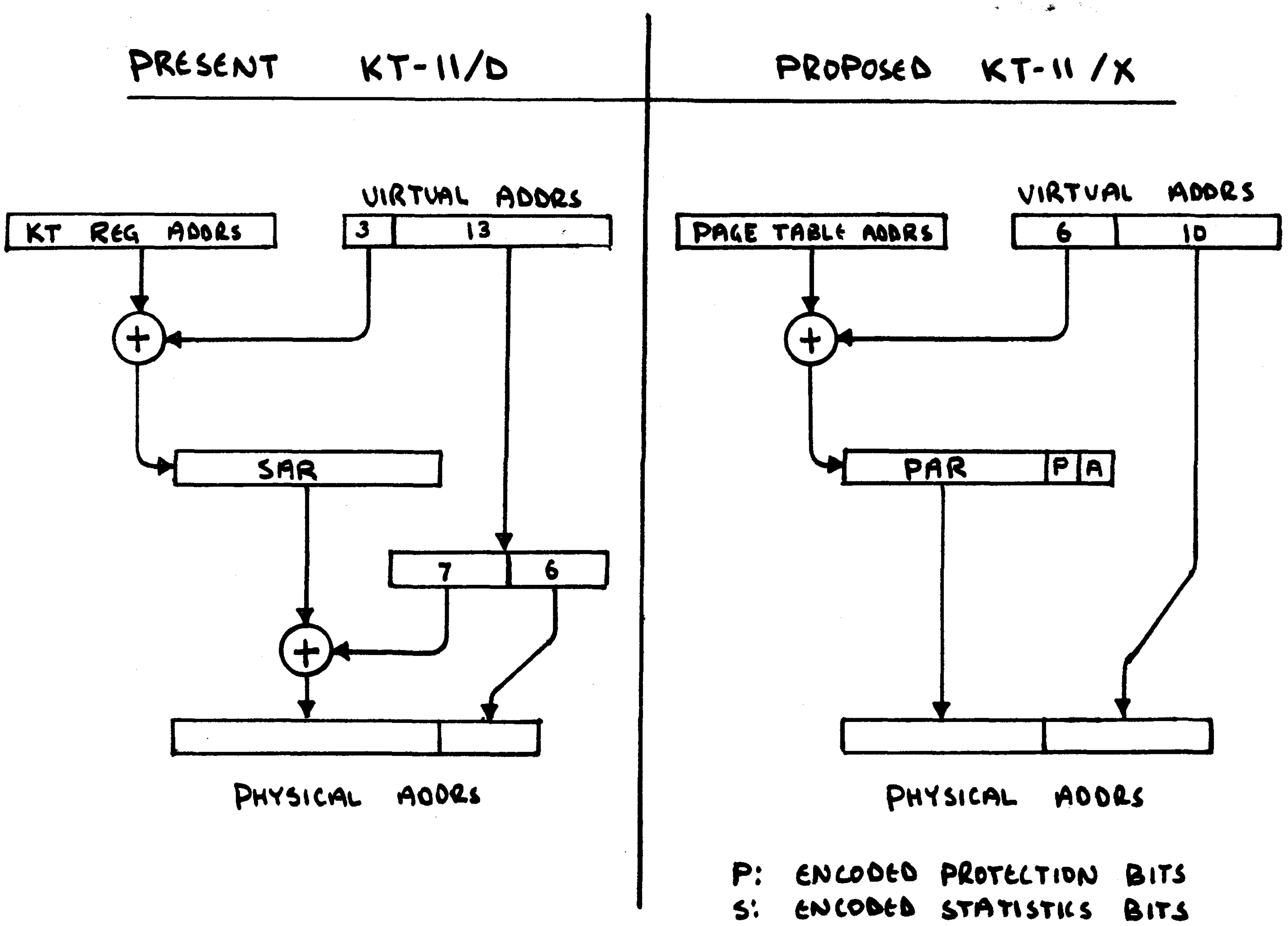

PRESENT KT-11/D
PROPOSED KT-11/X
VIRTUAL ADDRS
KT REG ADDRS
3
13
SAR
7
6
PHYSICAL ADDRS
VIRTUAL ADDRS
PAGE TABLE ADDRS
6
10
PAR
P
A
PHYSICAL ADDRS
P: ENCODED PROTECTION BITS
S: ENCODED STATISTICS BITS

fig III.1

SECTION IV

VIRTUAL ADDRESS SPACE EXTENSION

IV. VIRTUAL ADDRESS SPACE EXTENSION

There is little doubt that a major restriction of the PDP-11 is its limitation of program size to 32K words.

Competively, the PDP-11 does not compare favorably as evidenced by the following chart:

	HP	DG	MOD IV	INT	IBM	PDP-11
Virtual address space (bytes)	96K(1)	64K	256K(2)	2^{24}	2^{24}	64K(3)
I&D separation	Y	N	Y	N	N	Y
Physical address space	128K	256K	512K	2^{21}	2^{24}	2^{22}

1. Program segments restricted to 32K, data segments to 65K. Operating system features segmentation which allows subroutine segments to coexist in separate address spaces.

2. Includes factor of two for I&D space separation. Note that in practice, this is not attainable.

3. Excludes I&D space separation because it is unsupported.

This section contains descriptions of two approaches to virtual address space extension: segmented and linear.

SECTION IV.1

LINEAR ADDRESS SPACE SPECIFICATION

(To Be Included)

IV.2 Segmented VAX Overview

Attached is a preliminary specification for extending virtual address space for the PDP-11. The concepts and content of this document was, to a large extent, extracted from previous work done by Craig Mudge and Bill Strecker, and to some extent incorporates the combined advantages of both proposals.

It should be clear that this is a specification for an architecture, and not an implementation. Some freedom has therefore been exercised in creating two segment modes (1 KT compatible, and another with improved paging characteristics) that may be implemented as either in combined form, as seperate options, or excluding one or the other entirely. The combined architecture serve to show their comparisons and differences, and also serves to introduce compatibility whenever required.

The assumptions used in designing the extension logic were those presented in the September 13th memo from Craig Mudge, with some relaxation on the assumption of KT-11 compatability. Accordingly, some attempt has been made to clarify the motivation and advantages of a smaller page size than that presently used on the KT. Additional motivation for the smaller page size is the good possibility of the paging architecture being applied to small 11/05 systems, whereby protection would be implemented by a single segment*page table, thereby not only allowing for better protection on RT11 size systems, but also providing the ability for these small systems to run within a segment on the larger 32bit machines.

The total effort involved in extending the address space of the PDP-11 extends far beyond the effort required to design and build the basic machine. Effects on operating systems, compilers and linkers are significant if implemented in their full potential. The section dealing with these effects and related implications of the full implementation will be presented later.

* Previously called 'chapter' by Mudge, 'segment' by Strecker. The term segment is not consistent with KT11 documentation by more consistent with usage in the computer industry.

L Nelson

PDP 11/VX SPECIFICATION

Overview

The PDP 11 architecture explicitly involves the use of the general registers for all memory reference address modes. This characteristic suggests a natural means for extending the virtual address ability of the machine by simply extending the width of each register to say 32 bits. The concatenation of each register with its extension forms a 32 bit segmented address space with a logical capability of 2^{16} segments each having a length of 2^{16} bytes. The PDP 11/VX central processor is capable of executing existing programs residing within a segment with additional instructions provided for control and data transfer among segments.

The translation of the virtual address to the physical address is managed by the contents of segment tables which point to page tables whose contents in turn point to memory locations. This clasical segmentation /paging architecture combines the advantages of program construction and protection afforded by virtual address space segmentation, as well as the advantages of physical memory allocation and demand paging. The PDP 11/VX architecture allows for both KT11 compatible segments of lengths up to 8K bytes, as well as segments having a smaller fixed size of 1K bytes.

Appropriate mode bits have been added in the processor status register (PS) to provide compatibility with existing software particular concern has been given to interrupt and trap sequences, as well as segment and page fault recovery techniques. (to be supplied)

Correlations between expected functional performance/capability and related implications on the operating systems, compilers, linkers and additional hardware requirements are analyzed (to be supplied).

Formation of the Virtual Address

Since all PDP-11 memory reference instructions specify one of the general registers, the virtual address is formed by concatenating the contents of the extended register with the conventional PDP-11 effective address. Referencing figure 1, the extended register contents specifies the segment number and the PDP-11 effective address specifies the offset within the segment.

The address formation can be expressed more formally by using the following notation:

R = 16 bit general register as in the current PDP-11

RS = 16 bit extension of R called the segment register

▭ = concatenation operator

Re = Rs ▭ R
  = 32 bit extended register

( ) = contents of

The operation of the eight addressing modes is as follows:

<u>mode</u>	<u>operand address A</u>
Ø	R contains operand
1	A = (Re)
2	A = (Re) (R) = (R) + Δ

3	A = (Rs) ▭ ((Re))
	R = R + Δ

4	(R) = (R) - Δ A = (Re)
5	(R) = (R) - Δ A = (Rs) ▭ ((Re))
6	A = (Rs) ▭ (R) + X
7	A = (Rs) ▭ ((Rs) ▭ (R) +X)

Existing PDP-11 instructions will provide processing of data and control of programs within a segment. Additional instructions are defined to provide data transfer and control among segments.

1. Load Long (LL)
   opcode: Ø7 5R SS
   operation: The 2 word starting at the location specified by SS are used to load the extended register specified by P.

   The word pointed to by SS is used to load the segment register and the next word is used to load the general register. If SS is of the form ØR (i.e. mode Ø) then the extended register specified by R is loaded with contents of the extended register specified by R'. The value of Δ for autoincrement modes ls 4. In immediate mode the next two words inline with the next two words inline with the LL instruction are used to load the extended register.

2. Store Long (SL)
   opcode: Ø7 6R DD
   operation: The extended register specified by R is loaded into the 2 words starting at the location specified by SS. The format of the 2 words is as in the LL instruction. If DD is of the form ØR' then the extended register specified by R' is loaded with the extended register specified by R. In autoincrement and autodecrement modes the value of Δ is 4.

3. Jump to Subroutine Long (JSL)
   opcode: 10 7R DD
   operation: The 2 word destination specified by DD is saved in an internal register. (If DD is of form ØR an illegal instruction trap occurs.) Two words are pushed on the stack. The first is the low 16 bits of the extended register specified by R, The second word contains the segment register specified by R. The contents of the register specified by R are replaced by the contents of the (extended) PC. The contents of the PC are then replaced by the contents of the internal register.

4. Return from Subroutine Long (RSL)
   opcode: ØØ Ø2 1R
   operation: The contents of the (extended) PC are replaced by the contents of the extended register specified by R. The contents of the extended register specified by R are replaced by the top two words popped from the stack. The first word popped replaces the contents of the segment register specified by R; The second word popped replace the contents of the general register specified by R.

5. Jump Long (JL)
   opcode:
   operation: The contents of the PC are replaced by the contents of the double word addressed by DD.

Virtual to Physical Address Translation

There exists two independent motivations for the design of the address translation logic:

(1) First, the desirability for system and application programs to be logically divided into independent segments whereby control over protection, linking, sharing and executability can be affected at the segment level irrespective of the manner in which physical memory is allocated and associated.

(2) Second, the desirability for the operating system to physically divide memory into small pages whereby control over memory allocation, mapping and replacement (demand paging or virtual memory) can be affected at the physical page level irrespective of the manner in which the system and application programs are segmented.

Thus, segmentation relates to virtual address management, while paging relates to physical address management. The requirement that they co exist within the same architecture has determined the translation logic described herein.

To a large extent, the design of the KT-11 memory management unit does not reflect the above motivations (the 4K page size is too large, and the 32 word block size is too small). Nevertheless, architectural compatibility has been retained while at the same time, an operational mode (called page mode) has been provided which adequately meets the above motivations, thereby eliminating several KT deficiencies. This mode allows for more efficient allocation of memory and provides the possiblity of implementing demand paging (virtual memory) by reducing the page size to 512 words. Since the mode distinctions are applied at the segment level, both KT and page mode segments can be combined within a process running on a fully implemented system. Compatibility between the modes relating to access control, statistics gathering bits, etc., has been applied wherever possible.

Figure 1 has been sectioned into three areas to schematically describe the address translation mechanism: (1) the logic which currently exists on the PDP-11 with KT11 (enclosed in dashed lines), (2) the additional logic required for 32 bit segmented address (upper left), and (3) the additional logic to reduce the page size to that comensurate with the requirements to facilitate improved physical memory allocation and demand paging.

The translation of the physical address is accomplished by using the segment field as an index into a segment table (origined at a physical location specified by a base register, STBR). The segment table entry contains access bits, residency bit, KT mode bit, segment length, and a pointer to the origin of the segment's page table (origined on an eight word boundary).

The offset of the virtual address contains a page field (3 bits for KT mode, 6 bits for page mode) and a displacement. The page field is used as an index into a page table which contains statistics bits, a residency bit, and pointer to the physical page. The displacement field is used as an index into the physical page to locate the addressed word. The detailed format of each register is shown in figure 2.

Note that the contents of the segment tables relate to segment level access control, and the contents of the page table relates to physical memory management.

Note further that these table reside in physical memory, and are logically accessed for every memory reference. The actual number of core accesses per memory reference is solely a function of implementation, thereby satisfying the goal of common architecture implementable over a broad range of performance levels.

Compatiblity

In order to guarantee that existing user programs will run on the 11/VX machine (in an address space other than segment zero), a bit in the Program Status register, PS<08>, is assigned to specify X or non-X mode. When the X mode bit is set, the contents of all extended registers is taken from R7 (PC). This allows an extended program to call a non-extended subroutine by a normal JSR instruction.

To ensure that <u>system</u> programs are compatible, another spare bit PS <09>, is used to control the stacking and unstacking of PCX on interrupts and RTI, return from interupt. All interrupts are returned to Process Ø, Segment Ø. The interrupt vector here, in particular PS <09>, controls the stacking and unstacking of PCX.

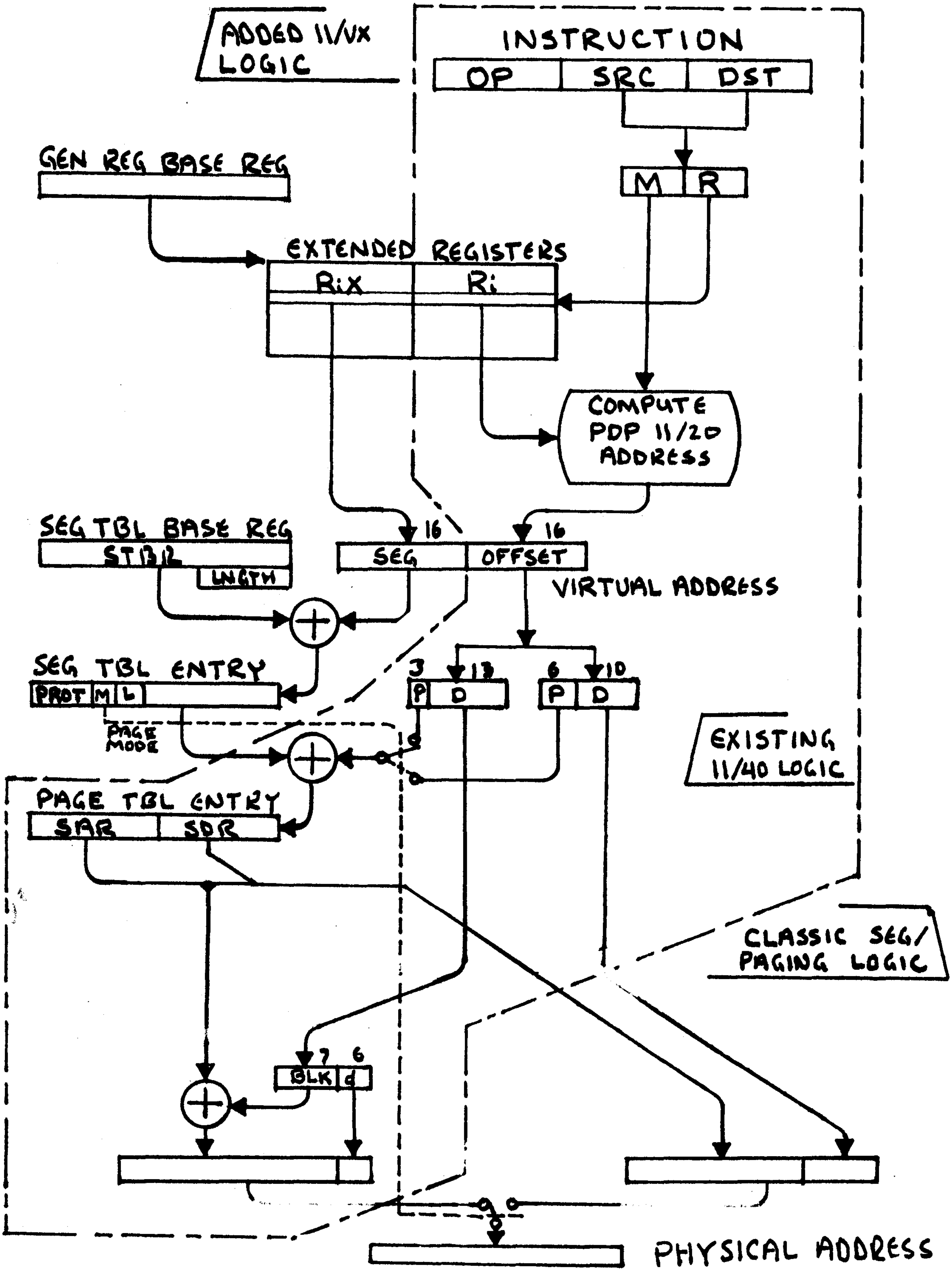

ADDED 11/VX LOGIC
INSTRUCTION
OP
SRC
DST
M
R
GEN REG BASE REG
EXTENDED REGISTERS
Rix
Ri
COMPUTE PDP 11/20 ADDRESS
SEG TBL BASE REG
STBR
LNGTH
16
16
SEG
OFFSET
VIRTUAL ADDRESS
SEG TBL ENTRY
PROT
M
L
PAGE MODE
3
13
P
D
6
10
P
D
EXISTING 11/40 LOGIC
PAGE TBL ENTRY
SAR
SDR
CLASSIC SEG/ PAGING LOGIC
7
6
BLK
d
PHYSICAL ADDRESS

Figure 1

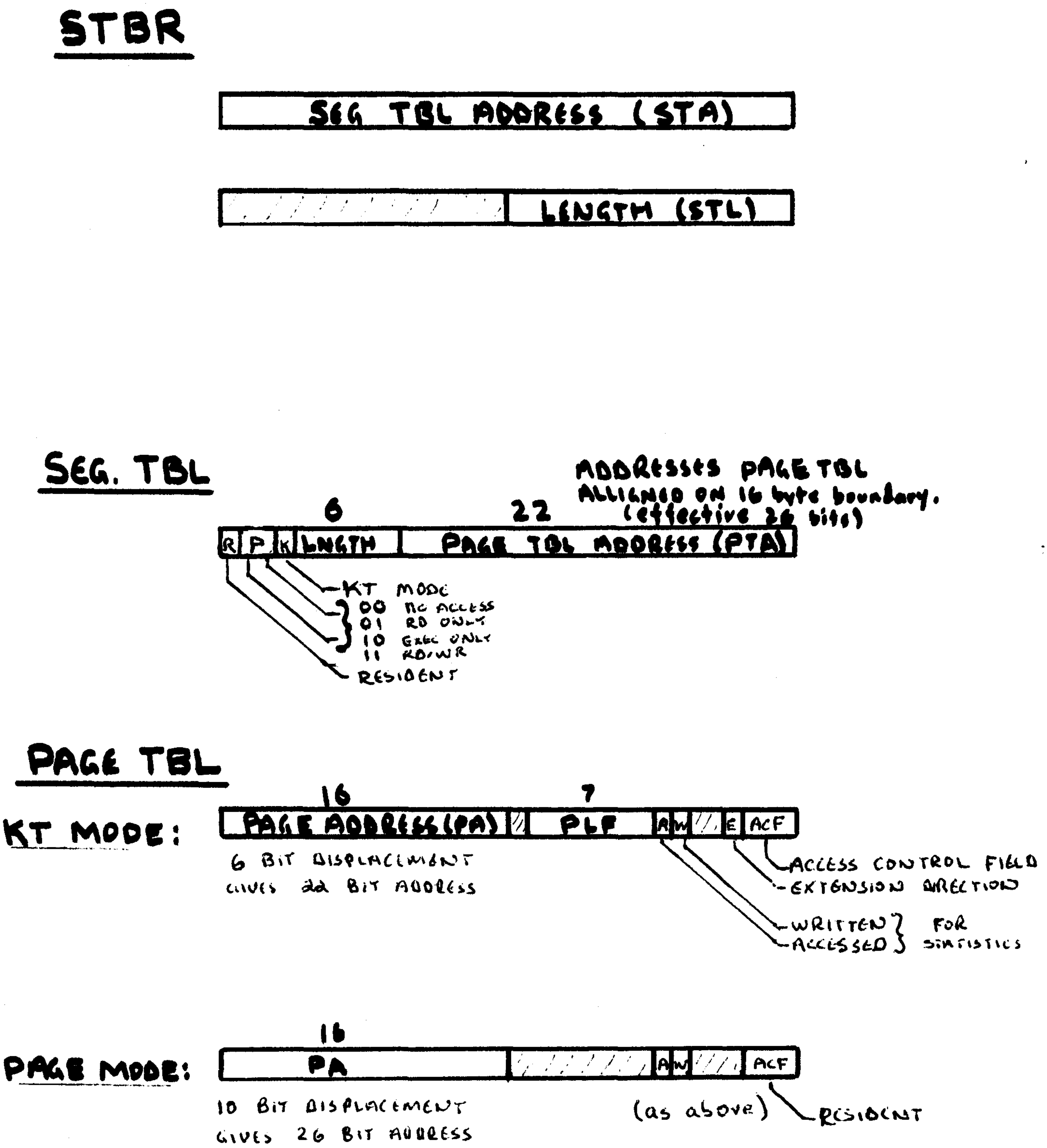

STBR
SEG TBL ADDRESS (STA)
LENGTH (STL)
SEG. TBL
ADDRESSES PAGE TBL
ALLIGNED ON 16 byte boundary.
(effective 26 bits)
6
22
R
P
K
LNGTH
PAGE TBL ADDRESS (PTA)
KT MODE
00 NO ACCESS
01 RD ONLY
10 EXEC ONLY
11 RD/WR
RESIDENT
PAGE TBL
KT MODE:
16
7
PAGE ADDRESS (PA)
PLF
A
W
E
ACF
6 BIT DISPLACEMENT
GIVES 22 BIT ADDRESS
ACCESS CONTROL FIELD
EXTENSION DIRECTION
WRITTEN
ACCESSED
FOR STATISTICS
PAGE MODE:
16
PA
A
W
ACF
10 BIT DISPLACEMENT
GIVES 26 BIT ADDRESS
(as above)
RESIDENT

Figure 2.

SECTION V

PDP-11 INSTRUCTION SET

(To Be Included)

SECTION VI

WRITABLE CONTROL STORE

(To Be Included)

SECTION VII

MULTIPROCESSING CONFIGURATIONS

VII. MULTIPROCESSING CONFIGURATION

Preliminary Price/Performance Analysis For Multiprocessor Configurations

Preliminary analysis shows that it makes more sense to consider tightly coupled multiprocessor systems to be based on large PDP-11/40 and PDP-11/45 systems rather than small PDP-11/05 systems. The reasons for this involve the cost of the MC11 and DA11 with respect to the cost of the processor, and the fact that a large number of processors will likely require memory configurations exceeding 28K.

Both the PDP-11/40 and the PDP-11/45 processors are sufficiently fast so that we can assume that the primary limitation in processing rate is imposed by the speed of the UNIBUS accessing memory. Therefore, if we assume that the processing rate is directly proportional to the rate of memory accesses, and that each processor randomly accesses each of M memory banks (true for interleaved memories), then the effective throughput of a system comprised of N processors can be shown to be:[1]

$$(1) \qquad \text{Throughput} = M(1-(1-\frac{1}{M})^N)$$

This expression for system performance has been computed for several configurations and divided by system price to obtain price/performance values. I have attached several figures which graphically represent these results. Figure 1 is a plot of Equation 1 showing lines of constant throughput ratio as a function of the number of processors and the number of parallel memory banks. Figure 2 shows the system throughput (in units of single processors) as a function of the number of memory banks, indicating that in configurations where the number of processors equals the number of memories, the system throughput is approximately 60% that of the theoretical maximum. Figure 3 shows a plot of the number of effective CPU's as a function of the number of actual CPU's for various memory configurations, operating in both bank and interleaved modes (Interleaved memory is accomplished by using the least significant memory address bits as bank addresses, whereas bank mode is accomplished by using most significant address bits as bank addresses. In the case of the former, memory addresses from processors are randomly distributed; whereas in the case of the latter each processor can execute a program which is local to each memory bank.). Figure 4 shows the price/performance ratio for various multiprocessor configurations as a function of typical single-processor system price. As is seen, large price/performance gains are realized in large system configurations. In Figure 5, we plotted the same price/performance data as a function of the total multiprocessor system price. Comparison of Figures 4 and 5 shows that for an eight-processor system, a factor of 2 increase in price performance is attained from a $300,000 system whose single-processor configuration would be priced at roughly $150K.

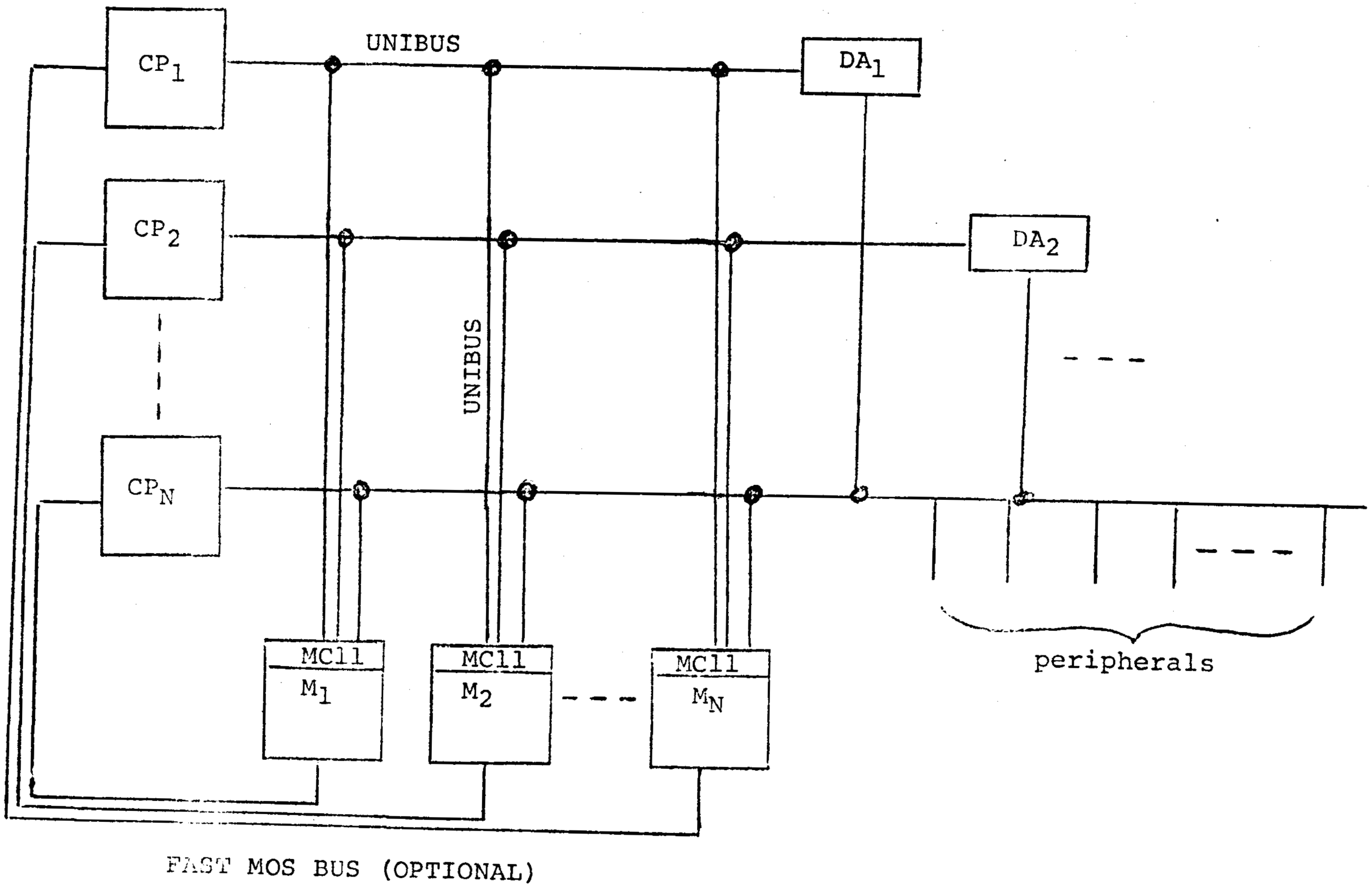
These price/performance calculations account for the current retail prices for PDP-11/40 systems, and for the current retail prices for the DA11-F and MC11. In addition, the performance degradation due to delays imposed by the MC11 is accounted for. Memory contention was computed under the assumption of random accesses which is a worst case assumption for interleaved memories. An additional 50% gain in price/performance can be realized by assuming that processors execute programs which are local to 16K banks.

A further significant increase in price/performance can be realized with the use of MOS memory. In this case, the scheduling algorithm would be designed to schedule particular processors to tasks which reside in that processor's high-speed memory, requiring that the scheduler be aware and take advantage of the PDP-11/45 high-speed memory bus. I would estimate that a 4 CPU system which utilizes this technique would have a price/performance ratio approaching 3:1 relative to a single-processor system selling in the area of $150K.

Regards.

DLN/ehb
attachments

[1] "Efficiency of a Multi-Control Path Processor," D. L. Nelson May, 1970 (report available)


UNIBUS
CP_1
DA_1
CP_2
DA_2
UNIBUS
CP_N
MC11
M_1
MC11
M_2
MC11
M_N
peripherals
FAST MOS BUS (OPTIONAL)

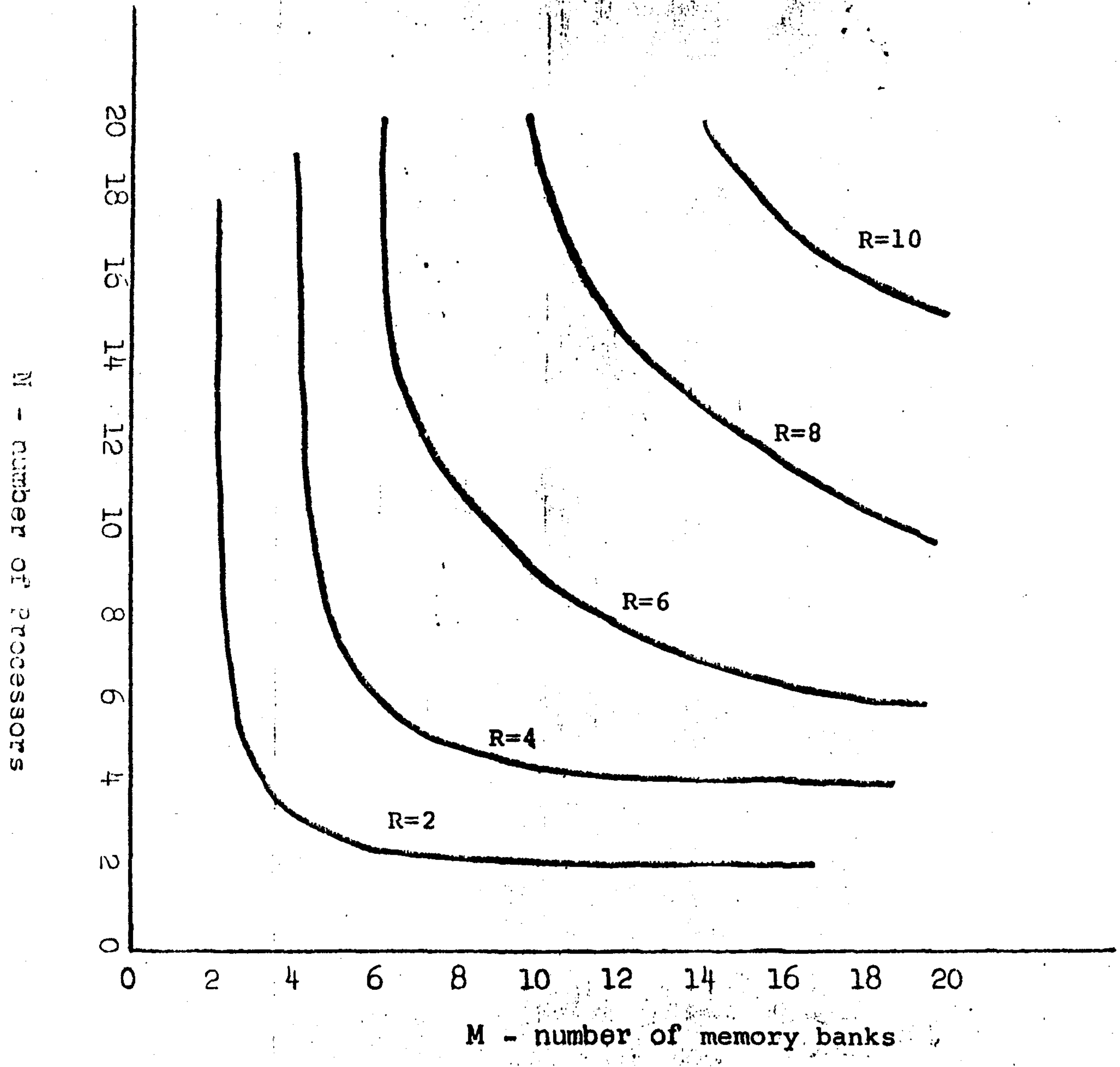


Figure 1. System Throughput Improvement Ratio

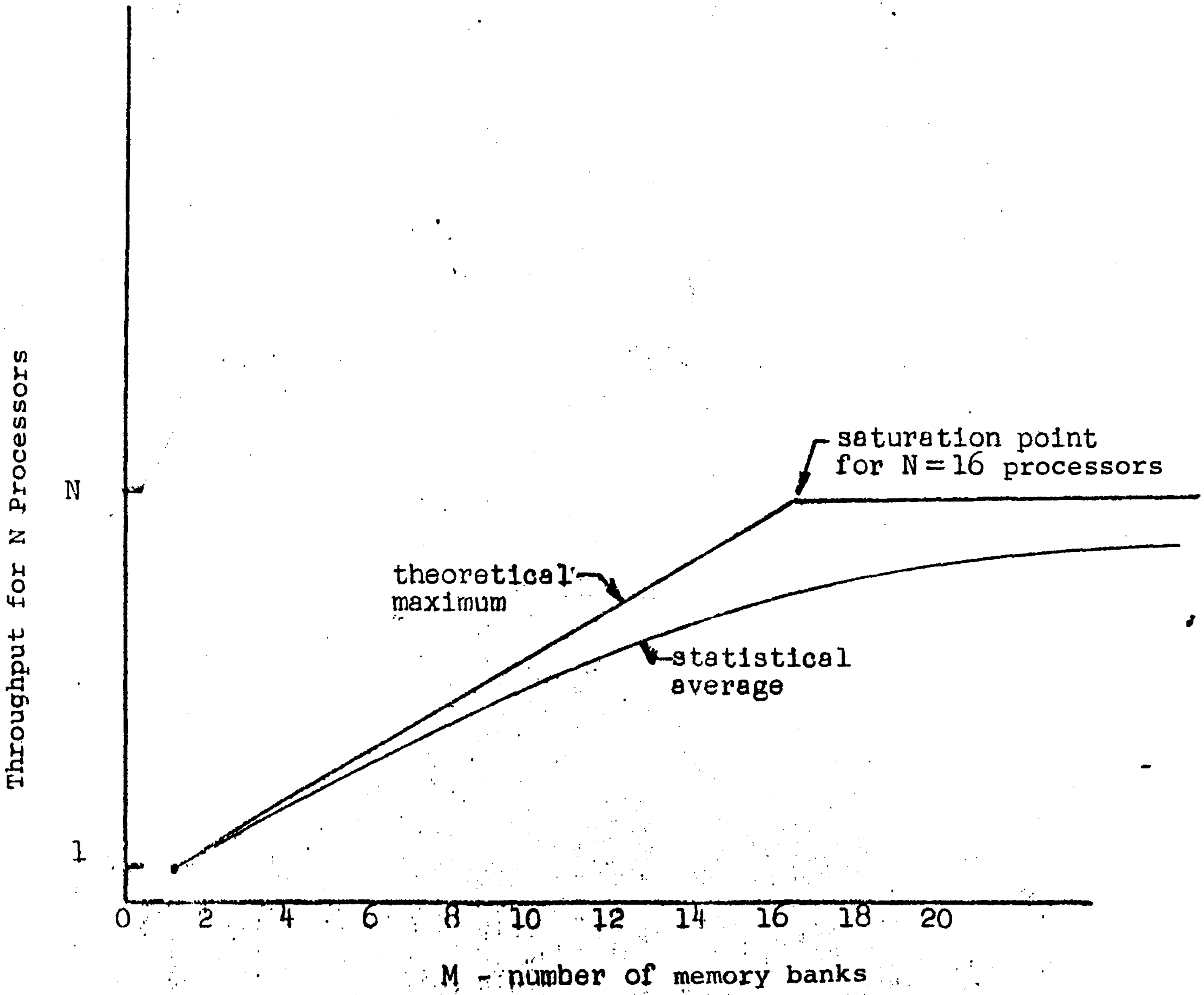


Figure 2. System Throughput

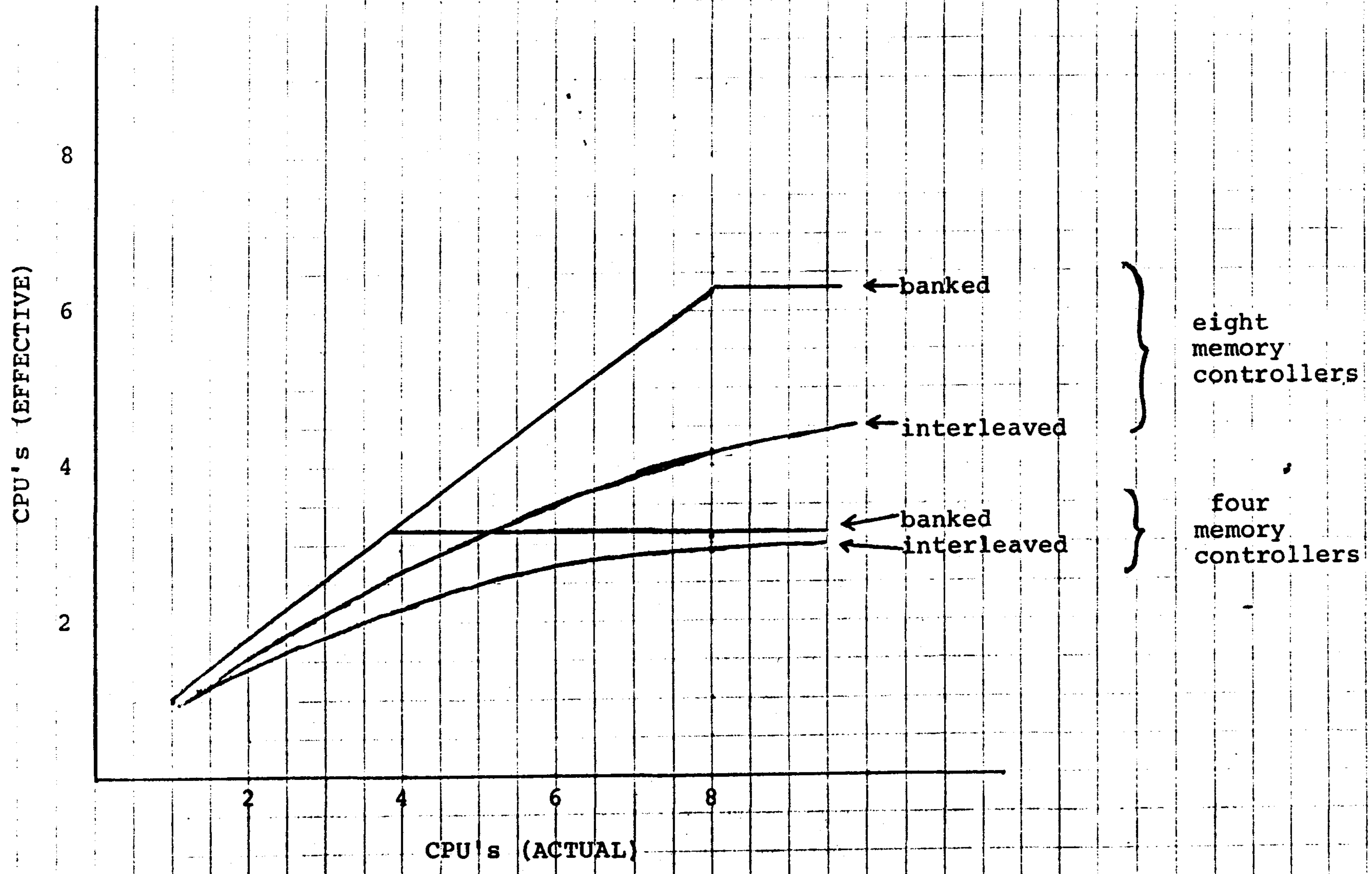

CPU's (EFFECTIVE)
8
6
4
2
2
4
6
8
CPU's (ACTUAL)
banked
interleaved
eight memory controllers
banked
interleaved
four memory controllers

Figure 3.

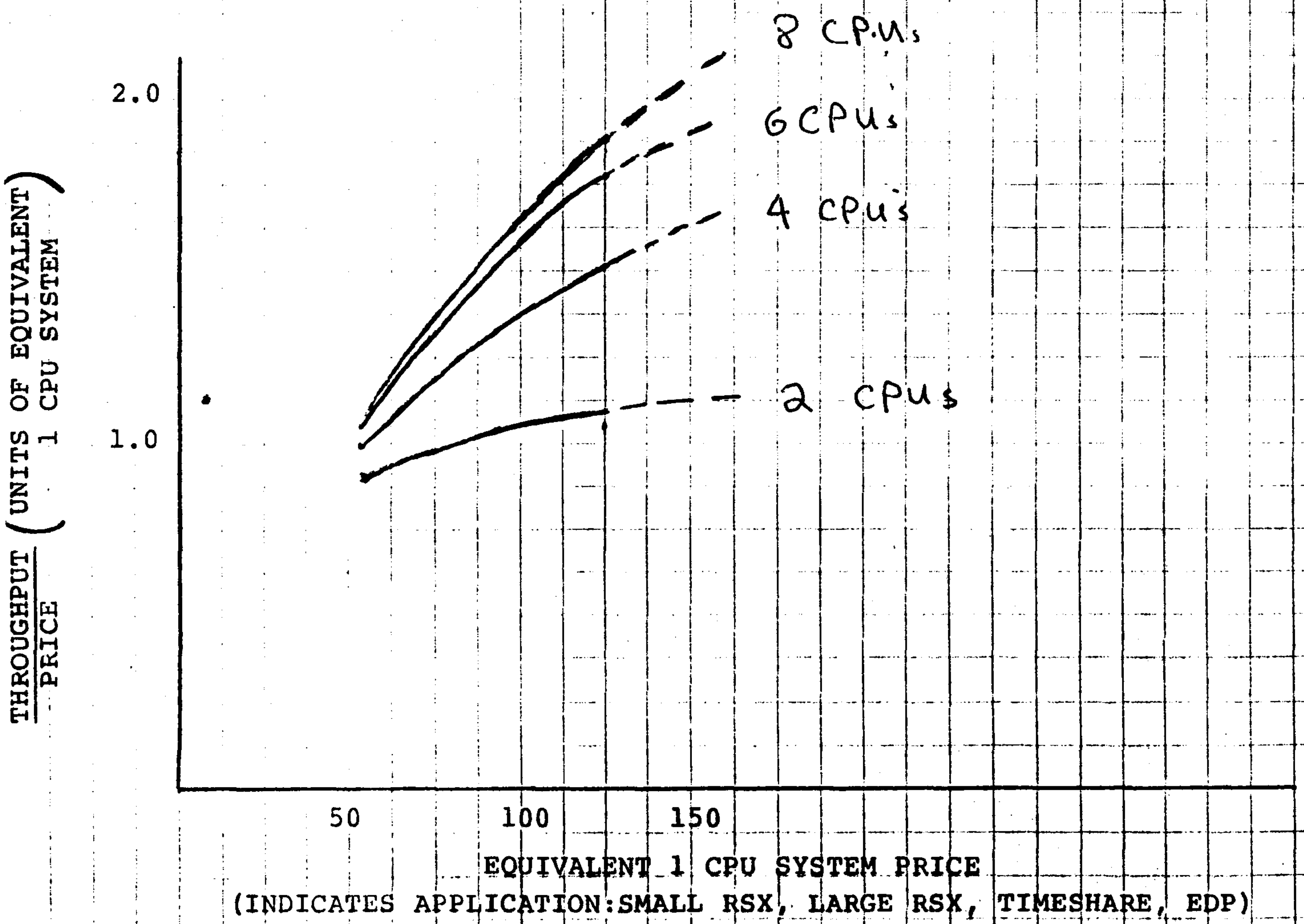

THROUGHPUT / PRICE (UNITS OF EQUIVALENT 1 CPU SYSTEM)
2.0
1.0
50
100
150
EQUIVALENT 1 CPU SYSTEM PRICE
(INDICATES APPLICATION:SMALL RSX, LARGE RSX, TIMESHARE, EDP)
8 CPUs
6 CPUs
4 CPUs
2 CPUs

Figure 4.

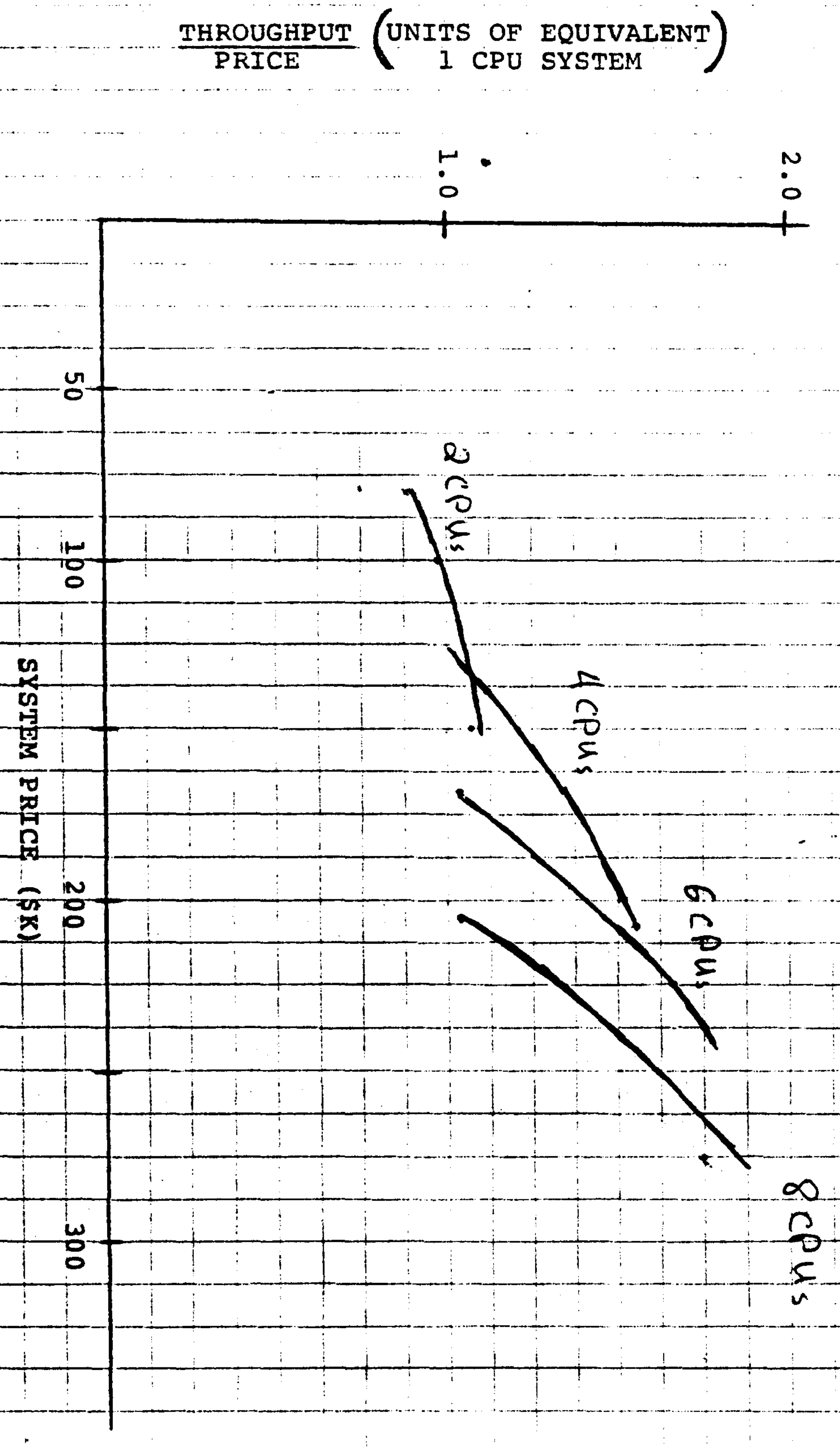

THROUGHPUT / PRICE (UNITS OF EQUIVALENT 1 CPU SYSTEM)
1.0
2.0
50
100
200
300
SYSTEM PRICE ($K)
2 CPUs
4 CPUs
6 CPUs
8 CPUs

Figure 5.

PDP-11 MULTIPROCESSING SOFTWARE PROTOTYPE
(CURSORY R&D PROJECT DESCRIPTION)

RICHARD H. ECKHOUSE, JR.
DECEMBER 13, 1974

MULTIPROCESSING WITH THE PDP-11 FAMILY COMPUTERS IS A NATURAL EXTENSION OF THE MULTIPROGRAMMING CAPABILITIES ALREADY PROVIDED BY RSX-11M AND RSX-11D. THE ADVANTAGES OF A MULTIPROCESSOR SYSTEM ARE MULTIFOLD. FIRST, IT ALLOWS THE USER TO INCREASE HIS TOTAL SYSTEM PERFORMANCE BY THE ADDITION OF ANOTHER PROCESSOR ELEMENT RATHER THAN BY REPLACING THE OLD SYSTEM WITH A NEW ONE. INDEED, IF THE USER ALREADY HAS THE LARGEST SINGLE PROCESSOR SYSTEM, INCREASING PERFORMANCE CAN ONLY BE ACHIEVED BY THE ADDITION OF ONE OR MORE PROCESSORS.

A SECOND ADVANTAGE OF THE MULTIPROCESSOR SYSTEM IS MODULARITY. BY MODULARITY IS MEANT THE ABILITY TO CONFIGURE SYSTEMS WHICH SPAN A WIDE RANGE OF CENTRAL PROCESSING PERFORMANCE WITH ONLY A SINGLE PROCESSOR DESIGN. DEC HAS ALREADY ACHIEVED SUCH MODULARITY IN ITS REAL-TIME SYSTEM SOFTWARE AND IT IS APPROPRIATE THAT IT DOES THE SAME THING FOR THE REAL-TIME HARDWARE SYSTEMS.

A THIRD ADVANTAGE IS THE REDUNDANCY IN THE CENTRAL PROCESSING CAPABILITY. SHOULD AT LEAST ONE OF THE PROCESSORS REMAIN OPERATIVE, THE FAILURE OF THE OTHERS WILL NOT CAUSE THE SYSTEM TO CEASE FUNCTIONING ENTIRELY. AVAILABILITY IS AN IMPORTANT ASPECT OF MOST OF THE MULTI-COMPUTER SYSTEMS WHICH DEC CURRENTLY SHIPS, AND A TRULY MULTIPROCESSING SYSTEM CAN BE EXPECTED TO PROVIDE SOME SORT OF "FAIL-SOFT" CAPABILITY.

THE HEART OF THE PDP-11 MULTIPROCESSOR SYSTEM IS SHARED MEMORY. SINCE THE PROCESSORS RUN IN AN ANONYMOUS MODE, SYSTEM FUNCTIONS ARE EXECUTED WITH EQUAL EASE BY EITHER PROCESSOR RUNNING REENTRANT PROGRAMS FROM THE COMMON MEMORY. AS MANY TASKS AS PROCESSORS MAY BE RUNNING SIMULTANEOUSLY, WITH EACH PROCESSOR HANDLING ITS OWN INTERRUPTS AND I/O PROCESSING. DUPLICATE COPIES OF SYSTEM PROGRAMS ARE NOT NEEDED SINCE PROCESSOR LOCKS HAVE BEEN ADDED SO AS TO PREVENT ONE PROCESSOR FROM INTERFERRING WITH ANOTHER WHILE CRITICAL SYSTEM TABLES ARE BEING MODIFIED. THESE LOCKS REQUIRE ONLY SMALL CHANGES TO THE STANDARD SOFTWARE SO THAT THE GROWTH IN EXECUTIVE SIZE IS NEGLIGIBLE.

USING COMMON TASK AND PARTITION DESCRIPTOR TABLES, THE INDIVIDUAL PROCESSORS WILL EACH BE ABLE TO EXECUTE ALL USER TASKS AND EXECUTIVE REQUESTS. ALTHOUGH I/O DEVICES WILL BE PHYSICALLY CONNECTED TO ONE PROCESSOR OR ANOTHER, THIS CONNECTION WILL BE TRANSPARENT TO THE USER TASKS, RESULTING IN THE LOGICAL SHARING OF ALL I/O DEVICES.

BY ADAPTING A CURRENTLY AVAILABLE SOFTWARE PRODUCT, RSX-11M, EXTENDED USER GROWTH IS ACHIEVED. EXTENDED GROWTH MEANS THAT THE SMALL USER CAN GROW FROM A SINGLE PROCESSOR

SYSTEM TO A MULTIPLE PROCESSOR SYSTEM BY A) BUYING THE REQUISITE HARDWARE AND B) REBUILDING RSX USING "SYSGEN". THE USER DOES NOT NEED TO LEARN A NEW SYSTEM OR NEW PROCEDURES BUT RATHER ADDS THE PROCESSORS IN MUCH THE SAME FASHION THAT HE WOULD ADD NEW I/O DEVICES.

HARDWARE REQUIREMENTS FOR THE MULTIPROCESSOR SYSTEM ARE RATHER MINIMAL. BESIDES THE ADDITIONAL PROCESSORS (UP TO 4 MAXIMUM), THE ONLY OTHER NEW PIECE OF EQUIPMENT ACTUALLY REQUIRED IS MULTIPORT MEMORY. THE SIZE OF THIS MEMORY IS DICTATED BY THE REQUIREMENTS OF RSX-11M (E.G., A MINIMUM OF 16K). SINCE ALL OF THE FEATURES SUPPORTED IN RSX-11M ARE SUPPORTED IN THE MULIPROCESSOR SYSTEM, THE USER CAN ADD MEMORY MANAGEMENT OR ADDITIONAL PERIPHERAL DEVICES IF HE SO DESIRES.

SECTION VIII

ASCII CONSOLE

VIII. ASCII CONSOLE

This section represents the collective efforts of the ASCII Console Committee represented by Small, Medium, Large 11 Engineering, 8 Engineering, and Field Service.

The Committee produced an architectural specification which all planned consoles will meet (11/05, 11/44*, 11/85, 8A*, Field Service). Each machine need implement only a subset of the specification (varying from all to none at all). The primary purpose of the specification, therefore, is to remove redundant implementations of common functions.

The Committee did not determine which functions should or should not be implemented on specific machines. Rather, this is a function of relevant product managers for the particular machines, and is based on marketing, cost, and implementation considerations that clearly lie outside of the Committees charter and domain.

Some specific issues, relating to consoles, which were not reconciled include:

A. Whether or not ASCII console machines should be capable of remote maintenance (11/05 will have serial consoles which cannot be controlled remotely).

B. Whether or not ASCII consoles should facilitate remote console control in a network for the purpose of remote program loading, etc.

C. The alternatives to the use of the ESC character in our present terminal oriented software.

Consequently, even though the Committee has successfully removed all apparent redundancies and has provided a consistent framework which relates to all known console functions, the Committee did not (nor did it try) to reconcile the diverse product philosophies that will continue to affect our strategies in the areas of networks, terminal oriented software, and remote maintenance.

* Cancelled

INTRODUCTION:

THE PURPOSE OF THIS SPECIFICATION IS TO DEFINE A SERIAL LINE SYNTAX AND A BASIC ASCII COMMAND SET FOR THE IMPLEMENTATION OF A CPU CONTROL DEVICE REPLACING THE LIGHTS AND SWITCHES CONTROL PANEL. THE RESERVED ASCII COMMANDS ARE INTENDED TO PROVIDE A CORE OF GENERIC OPERATIONS WHOSE PROPERTIES APPLY TO MOST DEC PROCESSORS. PROVISION IS MADE TO EXPAND THIS SET IN AN ORDERLY FASHION FOR EACH SPECIFIC CPU DESIGN. IT IS EXPECTED THAT EACH CONSOLE IMPLEMENTATION WILL PROVIDE A DESIGN DESCRIPTION THAT REFERENCES THE EXACT CPU REGISTERS INVOLVED AND ANY RESTRICTIONS THAT MAY APPLY, RATHER THAN THE GENERAL DESCRIPTION INCLUDED WITH THE COMMANDS IN THIS SPECIFICATION.

IN NORMAL OPERATION ASCII CONSOLE COMMANDS ARE MULTIPLEXED OVER A SERIAL LINE SHARED WITH PROGRAM I/O. THE PROTOCOL DEFINES THE SEQUENCE FOR ARBITRATING THE LINE MULTIPLEXOR VIA THE DATA STREAM (ALTERNATE MODE). A HARDWARE SWITCH MAY ALSO BE PROVIDED WHICH CAN DISABLE ARBITRATION AND FORCE THE DATA STREAM TO EITHER THE CONSOLE OR PROGRAM I/O CONTROLLER. THE SERIAL LINE PROTOCOL DEFINED HERE APPLIES ONLY TO EXCHANGES WITH THE ASCII CONSOLE LOGIC. THE CONSOLE APPEARS TRANSPARENT EXCEPT FOR THE ESCAPE SEQUENCE WHEN THE LINE IS IN ALTERNATE MODE.

1. SERIAL LINE PROTOCOL:

THE SERIAL LINE SYNTAX FOR CONSOLE COMMAND INPUT AND RESPONSE IS DEFINED IN THIS SECTION. THE PROTOCOL IS INDEPENDENT OF LINE SPEED OR FULL DUPLEX/HALF DUPLEX OPERATION. IN FULL DUPLEX OPERATION EACH CHARACTER IS ECHOED AS RECEIVED, EXCEPT WHERE DEFINED OTHERWISE BY THIS SPECIFICATION. IN HALF DUPLEX NO CHARACTER IS ECHOED.

1.1 COMMAND FORMAT:

THE COMMAND INPUT FORMAT IS AN ARBITRARY NUMBER OF OCTAL DIGITS TERMINATED BY A COMMAND CHARACTER OR COMMAND "SPECIAL SEQUENCE".

1.2 COMMAND RESPONSE:

THE COMMAND RESPONSE CONSISTS OF: THE ECHO OF THE COMMAND CHARACTER(S) (FULL DUPLEX LINE ONLY); THE TRANSLATION OF ANY NON-PRINTING COMMAND TO A PRINTABLE RESPONSE; THE RESPONSE DATA (AS REQUIRED); AND THE COMMAND ACKNOWLEDGE (ASCII 040).

NOTE: COMMAND ACKNOWLEDGE IS REPRESENTED BY THE HEART SYMBOL "♥".

MULTIPLE RESPONSE:

IF A SINGLE STIMULUS IS REQUIRED TO TRANSMIT SEVERAL RESPONSES (AN OUTPUT MACRO), THEN EACH SECONDARY RESPONSE MUST BE PRECEDED BY THE ASSOCIATED COMMAND CHARACTER ON EITHER A HD OR FD LINE. EXAMPLE: ON DETECTION OF A PROGRAMMED HALT THE NORMAL HALT RESPONSE MIGHT BE FOLLOWED BY A CPU STATUS REQUEST COMMAND (ST) AUTOMATICALLY.

```
    EX:  *H000010♥ST1244♥
```

1.2.1 OCTAL DATA OR NON-DESIGNATED ASCII CHARACTER:

THE OCTAL DIGITS 0-7 (ASCII 60-67) ARE USED FOR NUMERIC DATA ONLY. THE LOW-ORDER THREE BITS OF THESE CHARACTERS ARE SHIFTED INTO THE RIGHT END OF THE TEMPORARY DATA REGISTER. THE HIGH OCTAL DIGIT OF THIS REGISTER IS LOST.

ANY UNASSIGNED ASCII CHARACTER IS TREATED AS A "NO-OPERATION" BY THE CONSOLE LOGIC. NOP CHARACTERS MAY OCCUR ANYWHERE IN THE COMMAND INPUT STREAM.

1.2.2 COMMAND WITH NO DATA RESPONSE:

THE COMMAND IS ACKNOWLEDGED AT COMPLETION BY THE TRANSMISSION OF A SPACE CHARACTER (ASCII '40).

```
    EX:  3000L♥
```

1.2.3 COMMAND WITH DATA RESPONSE:

OCTAL DATA IS OUTPUT FOLLOWED BY THE COMMAND ACKNOWLEDGE AT COMPLETION.

```
    EX:  E012737♥
```

1.2.4 ILLEGAL COMMAND:

AN ILLEGAL OPERATION WILL RESULT IN THE TRANSMISSION OF A QUESTION MARK (ASCII 077) FOLLOWED BY THE COMMAND ACKNOWLEDGE.

```
    EX:  1000L?♡
```

1.2.5 RECEIVE LINE ERROR:

THE CHARACTER IN ERROR IS TRANSMITTED WITH CORRECT PARITY AND FRAMING (FULL DUPLEX ONLY), THEN A PLUS SIGN (ASCII 053) IS TRANSMITTED FOLLOWED BY THE ACKNOWLEDGE.

```
    EX:  1000@+♡    (FD LINE @=ERROR CHAR ECHO)
    EX:  1000L+♡    (HD LINE L=LOC COPY OF XMT CHAR)
```

2. RESERVED CHARACTERS:

THIS SPECIFICATION EXPLICITLY DEFINES THE COMMAND ACTION AND RESPONSE FOR CERTAIN OF THE ASCII CHARACTERS USED AS SINGLE-CHARACTER COMMANDS. CONSOLES IMPLEMENTED UNDER THE TERMS OF THIS SPECIFICATION MAY NOT USE THESE CHARACTERS FOR ANY OTHER CONTROL FUNCTION. IF A PARTICULAR COMMAND CANNOT BE IMPLEMENTED, THAT CHARACTER IS TO BE TREATED AS A "NO-OPERATION" (NOP) BY THAT CONSOLE. PROVISION FOR THE IMPLEMENTATION OF ADDITIONAL CPU-SPECIFIC COMMANDS IS MADE THROUGH THE USE OF THE DOLLAR SIGN (ASCII 044) COMMAND DEFINED BELOW. ALL OTHER SINGLE CHARACTER COMMANDS ARE IMPLICITLY RESERVED FOR FUTURE BASIC COMMAND SET EXPANSION.

2.1 ABBREVIATIONS:

2.1.1 ADDRESS REGISTER (AR):
REGISTER CONTAINING THE ADDRESS FOR START,EXAMINE AND DEPOSIT OPERATIONS. EX: PDP/11=BUS ADDRESS REGISTER, PDP/8=CP MEMORY ADDRESS REGISTER.

2.1.2 DATA DISPLAY LINES (DD):
INTERNAL REGISTERS OR MULTIPLEXORS DEFINED BY THE CPU DESIGN SPECIFICATION. EX: PDP 11/45 DISPLAY DATA MULTIPLEXOR OUTPUT.

2.1.3 DEPOSIT FLAG (DEP):
R/W STORE BIT THAT IS SET TO ONE TO INDICATE THE PREVIOUS COMMAND WAS A DEPOSIT OPERATION. USED TO CAUSE A DEPOSIT-STEP FOR SEQUENTIAL DEPOSIT OPERATIONS.

2.1.4 EFFECTIVE ADDRESS (EA):
THE EFFECTIVE ADDRESS IS THE CONTENTS OF THE ADDRESS REGISTER JUSTIFIED TO THE NEXT LOW-ORDER CPU STORAGE WORD. THE EFFECTIVE ADDRESS IS USED TO PERFORM START,DEPOSIT AND EXAMINE OPERATIONS.

2.1.5 EXAMINE FLAG (EXM):
R/W STORE BIT THAT IS SET TO ONE TO INDICATE THE PREVIOUS COMMAND WAS AN EXAMINE OPERATION. USED TO CAUSE AN EXAMINE-STEP FOR SEQUENTIAL EXAMINE OPERATIONS.

2.1.6 OCTAL FLAG (OCT):
R/W STORE BIT THAT IS SET TO ONE TO INDICATE THE LAST CHARACTER WAS AN OCTAL DATA CHARACTER. USED TO CLEAR THE OCTAL TYPE-IN REGISTER ON THE FIRST DATA CHARACTER FOLLOWING ANY USE OF THE REGISTER CONTENTS.

2.1.7 OPEN FLAG (OPN):
R/W STORE BIT THAT IS SET TO ONE TO INDICATE THAT A MEMORY OR REGISTER LOCATION HAS BEEN EXAMINED. USED TO PERFORM AN IMPLIED DEPOSIT OF USER INPUT DATA.

2.1.8 PROGRAM COUNTER (PC):
REGISTER THAT CONTAINS THE ADDRESS OF CURRENT PROGRAM EXECUTION. EX: PDP/11=CONTENTS OF R7, PDP/8=CONTENTS OF CP MEMORY ADDRESS REGISTER.

2.1.9 SERIAL OUT (SØ):
THE SERIAL LINE FROM THE ASCII CONSOLE TO THE CONTROLLING DEVICE. MAY BE A LOCAL TELEPRINTER OR REMOTE VIA DATA

SET.

2.1.10 SWITCH REGISTER (SW):
REGISTER USED TO DRIVE THE CPU SWITCH REGISTER LINES.

2.1.11 TEMPORARY DATA REGISTER (TMP):
REGISTER USED TO PACK OCTAL TYPE-INS. LOW-ORDER THREE BITS OF OCTAL DATA CHARACTERS ARE SHIFTED INTO THE RIGHT OCTAL POSITION AND THE LEFT OCTAL DATA POSITION IS LOST.

2.1.12 CONDITIONAL ACTION []:
THE ACTION ENCLOSED BY THE BRACKETS IS CONDITIONALLY EXECUTED DEPENDING ON SOME CONDITION SPECIFIED BY A NOTE.

2.1.13 CONTENTS OF LOCATION ():
INDICATES A REFERENCE TO THE CONTENTS OF THE REGISTER OR MEMORY LOCATION ENCLOSED BY THE PARENS.

2.1.14 TRANSFER DATA >>:
INDICATES THE DATA SOURCE ON THE LEFT IS TRANSFERRED TO THE DESTINATION ON THE RIGHT.

2.2 CPU CONTROL PRIMITIVES:

THE FOLLOWING COMMANDS ARE SENT TO THE ASCII CONSOLE TO CONTROL THE OPERATION OF THE PROCESSOR. THESE COMMANDS ARE INTENDED TO REPRODUCE THE LEVEL OF CONTROL PROVIDED BY THE LIGHTS AND SWITCHES CONTROL PANEL.

2.2.1 A(1Ø1) DISPLAY ADDRESS:

ADDRESS REGISTER CONTENTS ARE UNPACKED TO THE SERIAL OUT.

```
    (AR) >> SO
```

2.2.2 C(1Ø3) CONTINUE:

CAUSES CPU TO RESUME EXECUTING INSTRUCTIONS AT THE ADDRESS SPECIFIED BY THE PROGRAM COUNTER. UNCONDITIONALLY RESETS THE HALT FF TO PERMIT CONTINUOUS EXECUTION.

2.2.3 D(1Ø4) DEPOSIT:

THE CONTENTS OF THE TEMPORARY REGISTER ARE STORED IN THE EFFECTIVE ADDRESS REFERENCED BY THE ADDRESS REGISTER. THE SECOND AND SUCCESSIVE COMMANDS WILL DEPOSIT IN SEQUENTIAL LOCATIONS. IF THE CONTENTS OF THE TEMPORARY REGISTER ARE NOT ALTERED BY NEW OCTAL DATA, THE PREVIOUS CONTENTS WILL BE USED.

```
    [ EA+1 >> EA ] #1
    (TMP) >> EA
    Ø >> OCT;  Ø >> OPN;  Ø >> EXM;  1 >> DEP

    COND #1: DEP=1
```

2.2.4 E(1Ø5) EXAMINE:

THE CONTENTS OF THE EFFECTIVE ADDRESS REFERENCED BY THE ADDRESS REGISTER ARE UNPACKED TO THE SERIAL OUT. THE SECOND AND SUCCESSIVE EXAMINE COMMANDS WILL EXAMINE SEQUENTIAL LOCATIONS.

```
    [ EA+1 >> EA ] #1
    (EA) >> SO
    Ø >> OCT;  1 >> OPN;  1 >> EXM;  Ø >> DEP

    COND #1: EXM=1
```

2.2.5 H(11Ø) HALT:

CAUSES CPU TO STOP EXECUTING INSTRUCTIONS. WHEN HALT COMMAND IS COMPLETED THE CONTENTS OF THE PROGRAM COUNTER ARE UNPACKED TO THE SERIAL OUT.

```
    (PC) >> SO
    Ø >> OCT;  Ø >> OPN;  Ø >> EXM;  Ø >> DEP
```

2.2.6 I(111) INITIALIZE:

CAUSES A SYSTEM RESET. ANALOGOUS TO PDP/8 CLEAR OPERATION OR PDP/11 START WITH HALT SWITCH ON. FOLLOWING THE INITIALIZE THE PROGRAM COUNTER IS

UNPACKED TO THE SERIAL OUT.

```
    (PC) >> SO
    0 >> OCT;  0 >> OPN;  0 >> EXM;  0 >> DEP
```

2.2.7 L(114) LOAD ADDRESS:

LOADS THE CONTENTS OF THE TEMPORARY REGISTER INTO ADDRESS REGISTER.

```
    (TMP) >> AR
    0 >> OCT;  0 >> OPN;  0 >> EXM;  0 >> DEP
```

2.2.8 M(115) READ DATA DISPLAY:

THE STATE OF THE DATA DISPLAY REGISTER OR MULTIPLEXOR IS UNPACKED TO THE SERIAL OUT. THIS COMMAND PROVIDES A MEANS OF READING THE CONTENTS OF CPU ERROR OR OPERATIONAL INFORMATION REGISTERS WITH A SINGLE COMMAND CHARACTER.

```
    (DD) >> SO
```

2.2.9 N(116) EXECUTE NEXT INSTRUCTION:

CAUSES THE CPU TO EXECUTE A SINGLE INSTRUCTION AND THEN HALT. THE HALT FF IS FORCED SET BY THIS COMMAND. AT COMPLETION OF THE COMMAND THE CONTENTS OF THE PROGRAM COUNTER ARE OUTPUT TO THE SERIAL OUT.

```
    (PC) >> SO
    0 >> OCT;  0 >> OPN;  0 >> EXM;  0 >> DEP
```

2.2.10 R(122) READ SWITCH REGISTER:

SWITCH REGISTER CONTENTS ARE UNPACKED TO THE SERIAL OUT.

```
    (SW) >> SO
```

2.2.11 S(123) START:

CAUSES A SYSTEM RESET AND TRANSFERS THE CONTENTS OF THE ADDRESS REGISTER TO THE PROGRAM COUNTER. THIS COMMAND ALWAYS RESETS THE HALT FF TO PERMIT THE CPU TO BEGIN EXECUTING INSTRUCTIONS FOLLOWING THE RESET.

2.2.12 W(127) WRITE SWITCH REGISTER:

THE CONTENTS OF THE TEMPORARY REGISTER ARE TRANSFERRED TO THE SWITCH REGISTER.

```
    (TMP) >> SW
    0 >> OCT;  0 >> OPN;  0 >> EXM;  0 >> DEP
```

2.3 CPU MACRO COMMANDS:

THE FOLLOWING COMMANDS ARE CREATED BY COMBINING THE PRIMITIVE COMMANDS INTO SEQUENCES AND DEFINING A SINGLE CHARACTER TO INVOKE THE SEQUENCE. THESE MACRO'S MAY NOT BE DUPLICATED BY TRANSMITTING THE SAME SEQUENCE OVER THE SERIAL LINE. THE MACRO SET PROVIDES A HIGHER LEVEL SYNTAX SIMILAR TO THE ON-LINE DEBUGGING TECHNIQUE FOUND IN PDP/11 SYSTEM SOFTWARE. IN ADDITION, THE COMMANDS HAVE BEEN DEFINED IN SUCH A WAY THAT THE SHARING OF THE SERIAL LINE BETWEEN THE CONSOLE AND PROGRAM I/O IS MADE RELATIVELY TRANSPARENT TO THE USER.

2.3.1 G(107) GO:

CAUSES THE SERIAL LINE TO SWITCH TO PROGRAM I/O MODE, THEN STARTS PROGRAM EXECUTION AT THE ADDRESS IN THE TEMPORARY REGISTER. THIS COMMAND IS ANALOGOUS TO THE START PRIMITIVE.

```
    SEQ: Z, L, S
    0 >> OCT;  0 >> OPN;  0 >> EXM;  0 >> DEP
```

2.3.2 P(120) PROCEED:

CAUSES PROGRAM TO RESUME EXECUTION AND SWITCHES LINE TO PROGRAM I/O MODE. THE HALT SWITCH IS FORCED RESET TO PERMIT CONTINUOUS EXECUTION SIMILAR TO THE CONTINUE PRIMITIVE.

```
    SEQ: Z, C
    0 >> OCT;  0 >> OPN;  0 >> EXM;  0 >> DEP
```

2.3.3 /(057) OPEN:

EXAMINES THE LOCATION AT THE EFFECTIVE ADDRESS IN THE TEMPORARY REGISTER.

```
    SEQ: L, E
    0 >> OCT;  1 >> OPN;  1 >> EXM;  0 >> DEP
```

2.3.4 CR(015) CLOSE:

DEPOSITS ANY OCTAL DATA ENTERED SINCE THE LOCATION WAS OPENED, THEN CLOSES THE LOCATION. IF NO LOCATION IS OPEN THEN A LINE FEED IS SENT.

```
    [ D ] #1
    0 >> OCT;  0 >> OPN;  0 >> EXM;  0 >> DEP

    COND #1: OCT=1, OPN=1
```

2.3.5 LF(012) OPEN SEQUENTIAL LOCATION:

DEPOSITS ANY OCTAL DATA ENTERED SINCE THE LOCATION WAS OPENED, THEN CLOSES THE LOCATION. THE NEXT SEQUENTIAL LOCATION IS THEN OPENED AND THE ADDRESS AND CONTENTS ARE SENT TO THE SERIAL OUT.

```
    [ D ] #1,#2
```

```
     [ EA+1 >> EA ] #2
     [ A ] #2
     [ E ] #2
     Ø >> OCT;

     COND #1: OCT=1
     COND #2: OPN=1
```

2.3.6 ^(136) OPEN PREVIOUS LOCATION:

DEPOSITS ANY OCTAL DATA ENTERED SINCE THE LOCATION WAS OPENED, THEN CLOSES THE LOCATION. THE PREVIOUS SEQUENTIAL LOCATION IS THEN OPENED AND THE ADDRESS AND CONTENTS ARE SENT TO THE SERIAL OUT.

```
     [ D ] #1,#2
     [ EA-1 >> EA ] #2
     [ A ] #2
     [ E ] #2
     Ø >> OCT;

     COND #1: OCT=1
     COND #2: OPN=1
```

2.4 CONSOLE CONTROL COMMANDS:

THE FOLLOWING COMMANDS ARE USED TO CONTROL THE ASCII CONSOLE LOGIC. AS WITH THE CPU COMMAND SET, ALL UNIMPLEMENTED COMMANDS MUST BE TREATED AS NOP'S.

THE CONSOLE LOGIC MAY BE CONNECTED SIMULTANEOUSLY TO A LOCAL TELEPRINTER AND A COMMUNICATIONS DEVICE USED FOR REMOTE CONSOLE ACTIVITY. THE REMOTE/LOCAL SWITCH ENABLES EITHER PORT TO ORIGINATE CONSOLE COMMANDS EXCLUSIVELY.

2.4.1 $(044) SPECIAL SEQUENCE:

THE DOLLAR SIGN IS USED AS THE FIRST CHARACTER IN A TWO (OR MORE) CHARACTER SEQUENCE FOR PROCESSOR-DEPENDENT COMMANDS. THE COMPLETE SEQUENCE IS TREATED AS A SINGLE CONSOLE OR CPU COMMAND. ANY CONTROL FUNCTION NOT PROVIDED FOR EXPLICITLY IN THIS SPECIFICATION MUST BE IMPLEMENTED AS A SPECIAL SEQUENCE.

ANY CHARACTER EXCEPT THE NUMERICS (060-067) MAY BE USED (FOLLOWING THE $) TO DEFINE SPECIAL SEQUENCE COMMANDS. THE NUMERICS RETAIN THE SAME DEFININTION AS IN THE RESERVED CHARACTER SET AND MAY BE USED TO SUPPLY DATA FOR A SPECIAL SEQUENCE COMMAND.

EX: $T FOR READ CPU STATUS DISPLAY

2.4.2 @(100) CLEAR OCTAL INPUT:

THIS COMMAND IS USED TO CLEAR THE OCTAL TYPE-IN REGISTER. THERE IS NO RUB-OUT PROVISION AND THE ENTIRE INPUT NUMBER MUST BE RE-TYPED.

2.4.3 Z(132) SET SLI OPERATION:

SWITCHES THE ASCII DATA STREAM TO THE PROGRAM SERIAL LINE CONTROLLER.

2.4.4 ESC 0 (33,060) ESCAPE TO CONSOLE:

THE CONSOLE ESCAPE SEQUENCE SWITCHES THE ASCII DATA STREAM TO THE CONSOLE LOGIC. THE ESCAPE SEQUENCE CHARACTERS ARE NOT ECHOED AS RECEIVED (FD) AND ARE NOT PASSED TO THE PROGRAM. WHEN THE ESCAPE SEQUENCE IS COMPLETE AN "*CON" MESSAGE IS SENT TO THE LINE. IF THE CHARACTER FOLLOWING THE ESCAPE IS NOT THE CONSOLE SWITCH CHARACTER, THAN BOTH CHARACTERS ARE TRANSFERRED TO THE RUNNING PROGRAM IN A "BURST".

2.4.5 0-7 (60-67) OCTAL DATA:

THE OCTAL DIGITS ARE USED TO TRANSFER BINARY DATA OVER THE SERIAL LINE. THE THREE LOW-ORDER BITS OF INPUT DIGITS ARE SHIFTED TO THE LOW OCTAL DIGIT OF THE TEMPORARY REGISTER. THE HIGH OCTAL DIGIT IS LOST. CONSOLE RESPONSES ARE UNPACKED INTO OCTAL DIGITS FOR TRANSMISSION OVER THE SERIAL LINE.

2.4.6 ^C(003) INITIALIZE CONSOLE MODE:

CAUSES THE CONSOLE TO EXIT FROM "LOGIN" MODE OR "SPECIAL SEQUENCE" MODE. RETURNS ALL CONSOLE FLAGS TO A NORMALIZED STATE AND MAY BE DEFINED TO INITIALIZE CONSOLE CONTROL OPTION REGISTERS.

2.4.7 ^E(005) READ CPU ID:

CONSOLE RESPONDS WITH A UNIQUE ALPHA-NUMERIC SEQUENCE ASSIGNED TO THE PARTICULAR CONSOLE IMPLEMENTATION.

```
    EX: ^E001145/004503  -REPRESENTING AN 11/45 CPU
                          AND DATE= 74-323 (JULIAN)
```

2.4.8 ^L(014) SET LOGIN MODE:

THIS COMMAND LOGICALLY CONNECTS THE LOCAL TELEPRINTER SERIAL I/O TO THE REMOTE EIA INTERFACE. CHARACTERS MAY BE EXCHANGED BETWEEN THE LOCAL TELEPRINTER AND THE REMOTE SYSTEM. THE CONSOLE WILL IGNORE ALL CHARACTERS EXCHANGED OVER THE LINE UNTIL THE LINE MASTER SENDS THE INITIALIZE CONSOLE COMMAND TO EXIT LOGIN MODE.

2.4.9 ^T(024) TEST SHIFT REGISTER:

CONSOLE LOGIC TRANSMITS A SYNCH SEQUENCE CONSISTING OF THE COMMAND ECHO (FD LINE), A NULL CHARACTER AND A RUBOUT. THE SHIFT REGISTER OPERATION IS THEN VERIFIED BY TRANSMITTING A WORD OF ALL ZEROS FOLLOWED BY A WORD OF ALL ONES TO THE SERIAL OUT.

3. ERRORS:

THE CONSOLE LOGIC DETECTS TWO CLASSES OF ERROR: THOSE THAT RESULT FROM A USER COMMAND AND THOSE DETECTED BY THE CONSOLE LOGIC INDEPENDENT OF USER ACTIVITY.

3.1 COMMAND EXECUTION ERRORS:

COMMAND EXECUTION ERRORS ARE DETECTED IN CONSOLE MODE AND DO NOT ALTER THE STATE OF THE CPU. THE RESPONSE TO A COMMAND EXECUTION ERROR IS A SINGLE CHARACTER FOLLOWED BY A COMMAND ACKNOWLEDGE (ASCII Ø4Ø).

3.1.1 ?(Ø77) ILLEGAL:

COMMAND COULD NOT BE INITIATED BECAUSE OF A CPU OR CONSOLE CONDITION. EX: DEPOSIT WHILE CPU RUNNING OR THE INPUT OF NUMERIC 8 AS DATA.

3.1.2 +(Ø53) LINE ERROR:

COMMAND WAS RECEIVED WITH A FRAMING, OVERRUN OR PARITY ERROR AND ABORTED. CONSOLE DESIGN SPECIFICATION SHOULD DETAIL WHICH ERRORS ARE DETECTED.

3.1.3 #(Ø43) CPU RESPONSE TIME-OUT:

THE CONSOLE MUST PROVIDE AN INTERNAL TIME-OUT FOR ANY CPU COMMAND THAT STOPS CONSOLE ACTIVITY UNTIL THE CPU RESPONDS. IF THE CPU FAILS TO RESPOND A TIME-OUT OCCURS, THE COMMAND IS ABORTED AND THE CONSOLE RETURNS TO THE READY STATE.

3.2 OPERATION ERRORS:

THE CONSOLE LOGIC DETECTS CERTAIN ERRORS IN PROGRAM OR COMMUNICATION LINE OPERATION. THESE ERRORS RESULT IN A MESSAGE WHICH IS NORMALLY DIRECTED TO THE SERIAL LINE MASTER. ALL AUTOMATIC RESPONSES ARE PRECEDED WITH AN ASTERISK. THE SERIAL LINE MUST BE LEFT IN THE SAME STATE (CONSOLE/PROGRAM I/O) AS WHEN THE ERROR WAS DETECTED.

3.2.1 PROGRAMMED HALT:

IF THE PROGRAM EXECUTES A HALT, THE NORMAL HALT MESSAGE IS PREFIXED WITH AN ASTERISK AND SENT TO THE SERIAL OUT. THE MESSAGE GOES TO THE CONSOLE MASTER DETERMINED BY THE REMOTE/LOCAL SWITCH.

3.2.2 CARRIER LOST:

WHEN THE CONSOLE IS IN REMOTE OPERATION AND CARRIER HAS BEEN RECEIVED FROM A REMOTE STATION, LOSS OF CARRIER WILL FORCE THE LINE TO "LOGIN" MODE AND SEND A MESSAGE TO THE LOCAL TELEPRINTER. THE MESSAGE MUST BEGIN WITH "*CAR" AND BE TERMINATED BY AN ACKNOWLEDGE CODE (Ø4Ø). NON-SPACE CHARACTERS MAY BE APPENDED TO THE "*CAR" FOR CLARITY.

3.2.3 WATCH DOG TIMER ERROR:

THE CONSOLE LOGIC MAY CONTAIN A ONE-SHOT WHICH IS RETRIGGERED UNDER CPU PROGRAM CONTROL. THE TIME-OUT ERROR MUST BE ENABLED/DISABLED UNDER SERIAL LINE CONTROL. IF THE TIMER IS ENABLED AND THE RUNNING PROGRAM FAILS TO UPDATE, THEN AN ERROR MESSAGE IS SENT TO THE SERIAL LINE MASTER. THE MESSAGE MUST BEGIN WITH "*W" AND BE TERMINATED BY AN ACKNOWLEDGE CODE(Ø4Ø). ADDITIONAL NON-SPACE CHARACTERS MAY BE APPENDED TO THE "*W" FOR CLARITY. THE SERIAL LINE IS LEFT IN PROGRAM I/O MODE.

ENTERED: 11-19-74
BY: STEVE SKELTON

PROTOCOL SUMMARY 11-19-74

DESCRIPTION	CHAR	RESPONSE	<<<COMMENT>>>
CPU CONTROL PRIMITIVES:			
DISPLAY ADDRESS	A	A[A]♡	(AR) >> SO
CONTINUE	*C	C♡	RESUME EXECUTION
DEPOSIT	*D	[D]D♡	(TMP) >> EA
EXAMINE	*E	E[D]♡	(EA) >> SO
HALT	H	H[A]♡	(PC) >> SO
INITIALIZE CPU	*I	I[A]♡	(PC) >> SO; SYS RES
LOAD ADDRESS	*L	[A]L♡	(TMP) >> AR
READ DATA DISPLAY	M	M[D]♡	(DD) >> SO
NEXT	*N	N[A]♡	(PC) >> SO
READ SWITCH REGISTER	R	R[D]♡	(SW) >> SO
START	*S	S♡	(AR) >> PC
WRITE SWITCH REGISTER	W	[D]W♡	(TMP) >> SW
CPU CONTROL MACROS:			
GO	*G	[A]G♡,NL	Z, L, S, NL
PROCEED	*P	P♡,NL	Z, C, NL
OPEN LOCATION	*/	[A]/[D]♡	L, A, E
CLOSE LOCATION	*CR	NL♡	(TMP) >> EA; CLOSE
OPEN NEXT LOCATION	*LF	[A]/[D]♡	CR, LF OPN NXT LOC
OPEN PREV LOCATION	*^	[A]/[D]♡	CR, LF OPN PREV LOC
CONSOLE CONTROL COMMANDS:			
SPECIAL SEQUENCE	$	$()♡	()= CPU DEPENDENT COMD
CLEAR OCTAL DATA	@	@♡	Ø >> TMP; RE-TYPE NO.
SET PROG I/O MODE	Z	Z♡,NL	ASCII DATA TO PROG
SET CONSOLE MODE	ESCØ	NL,*CON♡	ASCII DATA TO CONSOLE
OCTAL DATA	Ø-7	Ø-7	DATA TO/FROM CONSOLE
INITIALIZE CONSOLE	ØØ3	^C♡	CLEAR CONSOLE
READ CPU ID	ØØ5	^E[N]/[M]♡	N=CP ID,M=REV DATE
SET LOGIN MODE	Ø14	NL,^L♡	LOC-TO-REM CONVERSATION
TEST SHIFT REGISTER	Ø24	^T[Ø]/[7]♡	
ERRORS:			
ILLEGAL	?	D?♡	ACTION ILLEGAL
LINE ERROR	+	@+♡	FRAME, PARITY, OVERRUN
RESPONSE TIME-OUT	#	D#♡	NO CPU RESPONSE
PROGRAM HALT		NL,*H[A]♡	
CARRIER LOST		NL,*CAR♡	REMOTE OPERATION ONLY
WATCH DOG TIMER ERROR		NL,*W♡	NO PROG ACTIVITY

NOTES 1) ♡=CONSOLE ACKNOWLEDGE (ASCII Ø4Ø)
2) [ ]=ADDRESS [A], OR DATA [D] EXCHANGE
3) NL=NEW LINE (CR,LF)
4) *=ILLEGAL WHEN CPU IS RUNNING