PAGINATION AND SEGMENTATION OF THE PDP-6 AND OTHER PROBLEMS IN MEMORY CONTROL

David L. Pehrson

Lawrence Radiation Laboratory, University of California

Livermore, California

ABSTRACT

Time sharing requires improved hardware techniques for the efficient use of core memory. A description is presented of a dynamic address-relocation system, which is being implemented on the PDP-6 central processors of the Octopus system. The motivation for the features of Pagination and Segmentation is emphasized.

INTRODUCTION

Core memory is typically one of the most expensive parts of any computing system. The potential requirements for core-memory facilities within a time-sharing environment may be greater by at least an order of magnitude than those for "traditional" systems. This[1] is because of the quantity of information which must be present in core storage when many users are simultaneously active in the system, and the frequency with which this information may be changing.

Pagination and segmentation are, basically, dynamic address-relocation techniques which allow efficient management of core-memory storage. Their importance is particularly apparent when applied to time-sharing systems, because they allow core-memory costs to be brought more closely into balance with other system costs.

At the Lawrence Radiation Laboratory we are incorporating extensive modifications in the PDP-6 central processors of the Octopus system to provide pagination and segmentation capabilities. The system that is currently being implemented has gone through several stages of evolution at the conceptual level during its development. This was largely the result of our learning process that came with attempting to model an operational software system within the constraints of a given set of hardware capabilities at the various stages.

This is another example of implementation, at a hardware level, of system functions originally delegated to software. Others are index registers, exponent arithmetic units for floating-point operations, and indirect addressing. Pagination and segmentation is conceptually far more complex. However, it points up the fact that engineers must become more software-oriented as programmers must become more hardware-oriented.

The importance of pagination and segmentation does not appear to [illegible] widely appreciated. Therefore, this discussion will consider the system capabilities that were gained at each stage of the evolution process, in an effort to bring out the importance of pagination and segmentation within the time-sharing environment.

SOME SYSTEM OBJECTIVES

Let us first consider some of the problems which exist in attempting to operate a system in a time-sharing mode, and from these define some objectives. Probably the most basic problem, and the one which motivated our initial efforts, is that of efficient use of core memory.

Time sharing implies the sharing of a single processor among many users for a single-processor system, or the sharing of a small number of processors if we have a multiprocessor system. Any particular code is executed during several short time periods interlaced with the time periods allocated to the other active codes in the system. These time periods may have a maximum duration on the order of 10 msec. Therefore, these codes must reside simultaneously in core to allow efficient switching among them.

No real difficulty would exist given a core-memory system of arbitrarily large size, since the entire code of each active user in the system could reside in core until execution was completed. However, when one considers the potential core requirements of several dozen users who require the use of compilers, assemblers, and executive codes — as well as their own codes — even 256K of core becomes entirely inadequate.

It is therefore advantageous for individual codes to give up core memory that contains information no longer relevant, so that it is available to other users. More generally, core memory should contain only information relevant to the current needs of the codes being executed by the system. Secondary storage, in our case the General Precision Disc File, should contain complete copies of all codes of which portions are in core. If the swapping of information back and forth from disc to core memory is done at a system level, it is invisible to the programmer/user. This is precisely the concept of <u>single-level storage</u>.

Another feature which can reduce core-memory requirements is to allow many users to share a single copy of a code. It is anticipated in the Octopus System that some codes, such as a Desk Calculator, will be used by many users simultaneously. Compilers and assemblers also fit into this category. Let us define this class of codes as *common routines*. It is important to note that these common routines must be written in execute-only form. A separate, unique data area will be associated with the common routine for each user. Common, execute-only routines must be guarded against accidental or malicious modification since the results may affect a large number of users.

Protection of the various users from each other is an absolute necessity. Program bugs will continue to exist and it is unlikely that they will be reduced in the time-share environment. The potential results of catastrophic system shutdown justify any protection mechanisms that can be included.

The last objective to be discussed here is less clearly defined but no less important. The value of any time-sharing system is ultimately dependent on the ability of programmers to use it. Therefore, any features added to the system to meet the above objectives must *minimize programming constraints* on the users.

ORIGINAL PDP-6 SYSTEM

The PDP-6 Central Processor as received from Digital Equipment Corp. contains limited facilities for time-share operation. To review, some of its characteristics are:

a) 18-bit addresses

b) Hardware address indexing

c) Indirect address capabilities to an arbitrary level

d) Two modes of operation:

   1. Executive mode

   2. User mode

e) Simple address relocation

All addressing in executive mode is absolute with no relocation. Interrupts in user mode trap to executive mode. Certain privileged instructions may be executed only in executive mode.

In user mode, limited address relocation and protection features exist. Input-output operation may take place in user mode at the option of executive-mode codes.

Core memory may contain the codes for a set of users, say, A, B, and C, providing the total does not exceed 2^{18} locations as shown in Fig. 1. Relocation consists of the ability to add a constant contained in a Relocation Register to all addresses generated within the central processor before they are placed on the memory-address bus. The constant is eight bits in length and is added into the significant bit positions providing offsets which are modulo 1024. The Relocation Register may be accessed only in executive mode.

The advantage of this simple relocation over none at all is that those codes which are not placed at the beginning of core memory (B and C in the example) do not have to be relocated with a loader program at the time they are brought into core. As far as B or C is concerned, addresses are generated under the assumption that they begin at location zero. The offset is provided at a hardware level.

A Protection Register exists which also contains an 8-bit constant that is compared against the high-order bits of all addresses generated within the processor prior to relocation. It therefore serves as a bound on the addresses which may be generated, and if exceeded will produce an interrupt to executive mode.

Simple relocation does not solve the core management problem for several reasons. For example, the core-memory contents for four users, A through D, may be distributed (as shown in Fig. 2) at some point in time. Another user, E, requests service. The unused core-memory space exceeds his requirements but no single unused block in itself is sufficient. Two alternatives exist: Load his code in two or more pieces requiring relocation at a software level, or rearrange the core-memory assignments of the existing core-memory occupants, A through D. Either alternative presents a significant overhead to the system.

No mechanism exists to allow partial codes in core to implement the single-level storage objective. Execution of single copies of common routines is possible only through the use of elaborate special coding techniques. A more complete discussion of the difficulties associated with common routines is included in another section of this paper.

PAGINATION

The limitations of simple relocation in the PDP-6 processors became apparent as the Octopus system evolved. The decision was made, therefore, to incorporate modifications to the PDP-6 processors to increase the efficiency of core-memory use. The first stage of the evolution assumed a system with pagination capabilities only.

The address space of 256K locations is divided into equal-sized pages, each of length equal to a power of 2. One page length was defined to be equal to 2^{10} = 1024 locations so that 256 pages uniquely address all of memory. Therefore, an address is specified as a page (given by the most significant eight address bits), and a location, or line, within the page relative to its b[illegible]ning (given by the least significant ten address bits).

Let the addresses generated by the processor be defined as virtual addresses, and physical addresses in memory as memory addresses. Pagination, then, is the mapping of pages in virtual address space into pages in memory address space. (See Fig. 3.)

Pages in virtual address space are contiguous and normally begin at location zero. The associated pages in memory address space are not contiguous in general. In fact, they may be located arbitrarily with respect to each other in memory address space since the relocation mapping is done at the page level. The location of a line within a page retains its relative position.

Pagination is implemented at a hardware level by substitution of the high-order eight bits in the 18-bit address as shown in Fig. 4. A separate Page Table, containing an entry for each page, must exist in core for each user who is active in the system. The origin of the correct Page Table, corresponding to the user who is active in the system, is contained in a hardware register, the Pagination Base Register. It is loaded in executive mode by a system program before the user is initiated.

The large majority of user codes require significantly fewer than 256K locations. When this is the case, the Page Table need only contain entries up to the maximum page number that will be specified in address space resulting in core-memory savings. The bound contained in the Pagination Base Register guards against addressing an undefined page in a Page Table that does not contain the full 256 entries.

Full responsibility for maintaining the Page Table in core memory is placed at the system level, since only it knows the actual distribution of pages without memory. The user does not need to be aware that pagination exists, or that the mapping is taking place. The ability to have Page Tables with less than 256 entries requires that the maximum page number be specified by the programmer, compiler, or assembler.

It is apparent in the implementation just described that every reference to memory by the central processor requires an overhead of an additional memory reference to the Page Table, degrading system performance excessively. Therefore, it was planned to provide a very high-speed access small memory containing 256 words as part of the hardware relocation system. Let us refer to this as the Paging Memory. Before processing of a user was started, the system would mark each word of this memory as being invalid. The eight-bit page numbers generated by the processor would now serve to address the memory. The first access to a page by the program would cause the hardware to recognize the fact that that page entry in the paging memory was marked invalid. Therefore, hardware sequencing would access the Page Table in core memory to obtain the relocation data. The data would also be placed in the high-speed paging memory. All subsequent accesses to that page, therefore, would find the relocation data in the high-speed paging memory, eliminating subsequent accesses to the Page Table in core. It was felt that access time to the paging memory should not exceed 80 nsec in order not to degrade system performance. This is approximately the time required by the PDP-6 to add the contents of the Relocation Register with simple relocation.

What has been gained by incorporating pagination? Obviously, it reduces the problem of core management outlined previously, since the code for a particular user may be distributed throughout core memory. The largest contiguous blocks of core memory required are now equal to a page, or 1024 locations.

A logical extension of pagination is that it enables implementation of the single-level storage concept. In the Page Table in Fig. 4, control bits may be associated with the relocation data for each page. One bit may be defined to indicate whether or not the corresponding page is actually resident in core. Reference to a page not in core will cause an interrupt to executive mode. Computation will be temporarily halted until the desired page has been obtained off disc by the system. Generally, bringing a new page into core requires removing some other page to make room for it. This requires non-trivial strategy at the system program level.

> "The mechanism of paging, when properly implemented, allows the operation of incompletely loaded programs; the supervisor need only retain in main memory the more active pages, thus making more effective use of high-speed storage."[1]

Three additional control bits in the Page Table are defined to enable the type of access to the page. They are defined:

A) Allow page to be executed.

B) Allow data to be read from the page.

C) Allow data to be written into the page.

Therefore, the type of access to any page may be limited. Attempted access which has not been allowed will trap to executive mode. The absence of any of these enable conditions will deny all access to the page.

The enable bits give protection in several areas. First, they allow pure-procedure (execute-only) codes to be written for common use with little danger of modification due to program bugs. Second, they can completely deny access to any page. Last, they can serve as valuable debugging aids for user codes, since program branches to data-only pages, or attempts to read or write into execute-only pages, are detected earlier.

It is important to emphasize, at this point, that no awareness of pagination has been required of the programmer, with one exception. The exception is that to make full use of the enable bits as a debugging aid, page usage must be declared. This will be done by compilers for codes written in higher-level languages but must be declared by the programmers for codes written at the machine code-assembly level. The core swapping which is continually taking place is not visible at the user level.

Let us now consider in greater detail, as noted earlier, the problem of execution of single copies of common routines by multiple users. Two general cases exist when considering a single common routine. Either the routine occupies the same locations in virtual address space for all users or it does not.

Consider the implications of the former with the extension that a user, designated A, uses a set of common routines. Also, let user B use some other set of common routines, a subset of which is also used by A. For each routine in the common subset to occupy the same virtual address space for each user, it is necessary that every potentially common routine have reserved for it a unique area in virtual address [illegible] This is because the mix of routines required by any user cannot be predicted and the assignment of two common routines to the same locations in virtual address space will eventually conflict when both are required simultaneously. It is apparent that unique assignments in virtual address space for all potentially common routines is impossible with a virtual address space of 2^{18} locations. Any mechanism for artificially expanding virtual address space only postpones the problem.

The alternate solution is to not assign common routines to unique locations in virtual address space. References to each common routine by a user are made directly by relocating these references with a Loader program. However, a new difficulty now arises. How does a particular common routine, S, communicate within itself? All references within S can only be relative since its actual location in virtual address space varies from user to user. This requires the use of additional address indexing and/or indirect addressing at a software level, decreasing program efficiency. Probably more important, constraints have been imposed upon the programmer. Also, all common routines must be linked together by the Loader before the computation can be started. At debug time, it is not probable that a new code will be entirely executed successfully. The overhead required by the Loader has been partially wasted. Therefore, it would be desirable to perform at execution time, the linking of common routines and sections of a user's code which were separately compiled, if there were a reasonable way of doing so.

In our case, the limitations of pagination by itself as well as economics entered into the decision to incorporate the additional features of segmentation, although pagination is certainly a step in the right direction. It appeared that only an active-circuit, flip-flop memory would meet the speed requirements of the pagination memory, the cost of which is not insignificant. In addition, modifications to the basic sequencing of the central processors are required to maintain continuity of codes that encounter the page-not-in-core condition.

The conclusion was that it was possible to obtain a system with considerably more flexibility and capability, particularly with respect to execution of common routines, by incorporating segmentation along with pagination. The cost did not appear to exceed that of pagination by itself, since the requirement of the high-speed pagination memory was eliminated. Additional system capabilities were gained, but let us temporarily postpone their discussion.

SEGMENTATION

Very generally, segmentation is used for the allocation of virtual address space, whereas pagination is used to the allocation of memory address space. Virtual address space is expanded in our system to 2^{22} locations and may be viewed as a two-dimensional structure. (See Fig. 5.) The most significant seven bits of a virtual address define a segment number which is treated as one dimension. The least significant 15 bits specify one of the 2^{15} locations within a segment and may be treated as the other dimension in virtual address space. Segments are individually paged via the pagination mechanism.

Let us examine a slightly simplified system, as shown in Fig. 6, essentially the one from which our final system is derived. The addresses directly generated by the processor are still hardware-limited to 18 bits. The least significant 15 bits of an address designate a location within a segment. The segment number is generated via an indexing process by use of the significant three address bits to select one of eight Address Base Registers. Address Base Registers contain a 7-bit segment number and some control information. Therefore, the 22-bit virtual address is composed of the 7-bit segment number from the selected Address Base Register and the least significant 15 bits of the 18-bit address generated by the processor. Address Base Register 0 has special significance and is called the Procedure Base Register.

In user mode, programmers have direct load and unload access to all Address Base Registers other than the Procedure Base Register so that their contents may be modified by running codes. Only system programs in executive mode may read or modify the Procedure Base Register directly. Therefore, if any code requires access to more than eight segments, the segment numbers contained in the Address Base Registers must be modified during run time by the code itself.

Let us restate the difference between pagination and segmentation at this point. Segmentation is used for the allocation of virtual address space. Segments, therefore, are visible to the programmer and require his awareness, at least for codes written at the machine-code level. This awareness is taken care of for him by compilers when higher-level languages are used.

Pagination is used for allocation of memory address space. In particular, memory address space is still limited to 2^{18} = 256K locations so that the pagination mapping of the 22-bit virtual address must ultimately produce an 18-bit address. Pagination is applied to segments individually, primarily for efficiency. This is so that relocation information need only be provided for those segments actually used by any code. Therefore relocation by pagination consists basically of a two-level lookup operation, first into a Descriptor Table (or segment table) to find the Page Table Origin for the particular segment, and then into a Page Table for the relocation information for the page sought. If Page Tables were provided for every possible segment, whether used or not, every user would require 128 Page Tables to defined in core memory.

Let us further look at pagination in the context of our system with segmentation. One Descriptor Table exists in core memory for each active user in the system with a maximum length of 128 locations. The memory address of the beginning of the Descriptor Table is given by the Descriptor Table Origin, contained in the Descriptor Base Register. This is a hardware register loaded by executive-mode system programs prior to initiation of a user program. The segment number, contained in the selected Address Base Register, designates the desired Descriptor Table entry relative to the origin.

Each entry in the Descriptor Table contains the following information:

A) Origin of corresponding Page Table;

B) Length of corresponding Page Table (bounds);

C) Whether or not corresponding Page Table actually is defined.

The Page Table Origin serves essentially the same purpose as did the Pagination Base Register in our earlier discussion of pagination. The more significant address bits in the 15-bit intrasegment virtual address define the page number which is used as an index into the selected Page Table. The actual relocation information to be substituted into the significant address bits of the originally generated 18-bit address is given by the Page Table entry.

To summarize what has been [illegible] so far:

A) The processor-generated 18-bit effective address has been expanded to a 22-bit virtual address via an Address Base Register by prefixing a segment number to the intrasegment address.

B) This 22-bit address has been relocated (by pagination) by a two-stage lookup process of core-memory mapping tables, first to the Descriptor Table and then to a Page Table.

C) The resultant relocation data in the Page Table was substituted in the original 18-bit address to produce the relocated 18-bit memory address.

The Descriptor Base Register contains the Descriptor Table Bounds so that the Descriptor Table need not contain the full 128 entries. Similarly, each defined Page Table need not be of maximum length (since — in general most segments are not of 32K length), so that a bound for each Page Table given in the Descriptor Table along with the Page Table Origin.

It should be apparent to the efficiency experts that system efficiency is not too good at this point. We have potentially made the requirement that every reference to core memory by the processor requires two additional memory references — first to a Descriptor Table and then to a Page Table, to perform the relocation mapping. The high-speed pagination-memory approach previously discussed is not the answer, since we now need to map 2^{22} locations, which would require pagination memories of $2^4 = 16$ times our previous requirements.

The approach that has been incorporated into the system is to provide a very small, high-speed associative memory. (See Fig. 7.) The basic approach is used also on the GE 645 and IBM 360-67 systems. In our system, an 8-word associative memory is used. The 7-bit segment number and the 5-bit page number are the input data which are simultaneously matched against the corresponding information in the associative memory. If a match is obtained, the remaining 18 bits of information in the word are gated out.

The output information is an image of the relocation data contained in the corresponding Page Table entry in core memory and is substituted in the original 18-bit address. The entire process requires the same amount of time as originally required to add the Relocation Register constant in the old system. If a match cannot be made, hardware cycles take place to access the Descriptor and Page Tables in core memory. However, the entry contained in the Page Table along with the segment and page numbers is now placed in the associative memory in the word that was accessed by the system longest ago. Therefore, the associative memory contains relocation data for the eight most recently accessed pages contained in one or more segments. It should be emphasized that this algorithm is implemented at a hardware level and does not place an additional overhead on the software system.

For flexibility and efficiency, four page sizes have been defined;

A) 2^7 = 128 words

B) 2^9 = 512 words

C) 2^{11} = 2048 words

D) 2^{13} = 8192 words

where the pages contained in any single segment are all of equal length. The small page size allows conservation of core memory when very short sub-routines are required, since a page is the smallest amount of core which be allocated. The largest page size, 8192 words, should see very limited use. It allows reference to 8 X 8192 or 64K core locations without extra cycling to continually bring new entries into the associative memory. This might be used for certain types of compilers, such as LISP, which must reference — essentially randomly — large portions of address space. Most codes will use either 512- or 2048-word pages. Information on the disc will be paged, at a software level, into 512- and 2048-word blocks.

Since the maximum number of pages contained in a segment is limited (by hardware) to $2^5 = 32$, segments composed of 128- or 512-word pages are actually shorter than the 32K maximum length. In particular, the segment length and number of pages are limited as shown in Table I for the four page sizes.

Page size for a given segment is defined by a pair of the control bits in the selected Address Base Register. The first time a segment is accessed, hardware sequencing sets these bits equal to corresponding information in the Descriptor Table. This is done to inhibit users from illegally modifying the defined page size in the Address Base Register during execution, since this would potentially enable them to access undefined virtual address space.

Instructions are always obtained from the segment specified by the Procedure Base Register (Address Base Register 0), the significant three bits of the 18-bit program counter being zero. On jump instructions, the contents of the Addr ss Base Register selected by the destination address are transferred to the Procedure Base Register (Address Base Register 0) and only the least significant [illegible] bits transferre [illegible] program counter. Therefore, any s[illegible]ment of code does <u>not</u> care what actual segment number is associated with it, since addresses generated at execution time reference the Address Base Registers.

The disassociation of the segment number from the block of information that comprises the segment is the essential characteristic which has been gained by use of the Address Base Registers. Restated, a block of information which is called a segment may be given different segment numbers for each of several users and therefore allow the segment to occupy a different portion of virtual address space for each of them. We have, thus, a straightforward way to allow sharing of common routines by multiple users.

This may be made more clear by considering a specific example, as outlined in Fig. 8. Let A and B be two users who share a subroutine S. Let A jump to S via Address Base Register 2, and B jump to S via Address Base Register 4. Also S may be defined as segment number 60 for A and as segment number 34 for B. However, after the jump to START (within S), addressing is consistent since the Procedure Base Register is used by both for addressing the instructions comprising S.

A corresponding problem exists when returning from a common routine since the segment number associated with the calling segment is unknown. Two approaches may be used. The first requires the use of a standard Address Base Register to be used for the return of control in a manner analogous to that currently used to link subroutines by use of an index register.

The other approach (which has been allowed for in our hardware) makes use of a pushdown list which is treated as a unique segment. A pair of PUSH and POP instructions has been added. The PUSH instruction places the contents of the program counter <u>and</u> the Procedure Base Register in the pushdown list and then replaces the contents of the program counter with the "transfer to" address. The POP instruction is just the inverse so that at completion of the POP instruction, the Procedure Base Register contains the same segment number which it contained before the original PUSH instruction was given.

Either of the above approaches may be used. However, the PUSH-POP instruction pair eliminates the need for a calling program to load another Address Base Register with the contents of the Procedure Base Register prior to initiating the jump.

Segment number 127 $\left[(177)_8 \right]$ is defined to be illegal in our system. This serves two purposes. The segment-number fields of all Address Base Registers are set to 127 by hardware when the Descriptor Base Register is loaded in executive mode. Therefore, failure to load an Address Base Register by a running code prior to its access will be trapped, furnishing protection against a class of inconsistencies.

The other use allows the assignment of segment numbers and the linking of segments to be done at execution time. This can be implemented by providing a table in core memory which is itself a separate segment, which contains segment numbers corresponding to all other segments referenced by a running code. The user code references this table to obtain the segment number to be placed in the Address Base Register. If the segment number has been defined by the system, operation proceeds normally. However, 127 will be placed in the table by the system program for all segments which have not yet been defined. Reference to any of these segments will be trapped at the time the attempt is made to load 127 into an Address Base Register. The system interrupt program will then make the required assignment.

Why bother with linking of segments at run time? First, it allows the execution of codes compiled in several parts, each part being treated as a segment without requiring that all parts be bound together by use of a conventional Loader program. The largest amount of code that must be bound together is the segment. Second, it allows codes to run without prior planning

Augmentation allows references to entry points in segments other than the one being executed to be treated as symbols at the software level. As noted in the segmentation discussion, external segments were essentially treated as symbols, since definition of the actual segment number was made by system programs at run time. However, the relative entry location within the segment must have been known to the calling segment. With augmentation, the entry point — as well as the segment number — can be defined by the system at run time. The linking segment referenced by the running code now contains external symbols, not just external segment names.

Why is this advantageous? Basically, it reduces the communication required among programmers and also the amount of system reloading that must take place when a common routine is modified by its originator. Since entry points are now referenced as symbols, changing the relative location of the entry point does not require the awareness of users.

SUMMARY OF SYSTEM CHARACTERISTICS

In summary, let us restate that segmentation is used for the allocation of virtual address space. Segments are therefore visible to the programmer. Segmentation allows:

a) Reasonable use of common routines by multiple users;

b) The dynamic linking of separately compiled portions of code;

c) The dynamic linking of additional subroutines and/or data areas not foreseen at the outset.

d) The ability to make external references by symbol name.

Pagination, on the other hand, is implemented entirely at the system level and provides efficient core management through implementation of the single-level storage concept.

The cost of pagination and segmentation systems for <u>both</u> PDP-6 processors is somewhat less than the cost of 16K of core memory. The effective savings in core-memory requirements (or alternately stated, the increase in system throughput for a given amount of core) cannot be accurately evaluated at the present time but is expected to be significant.

A major portion of the pagination and segmentation system must be implemented at the software level. It is expected that the software will go through several levels of sophistication as demand on the system increases and as knowledge of system characteristics is obtained.

IMPLEMENTATION AND CURRENT STATUS

Real time overhead attributable to relocation hardware has been kept to a minimum through use of very high-speed integrated circuits and logic design for minimum delay. The total relocation time is estimated to be approximately equal to that of the original PDP-6 system.

SUHL II (Sylvania Universal High Level) integrated circuit logic was used that has a stage propagation delay of six nsec, typically. This type of logic also features an active pull-up transistor in the output stage to minimize effects of capacitive wire loading. Four-layer printed-circuit boards were used to mount 27 and 32 integrated circuits in two cases, respectively, in arrays. The integrated circuits will be operated off ground to allow direct interfacing with DEC logic levels; thus the time delay attributable to level shifting will be avoided.

The pagination and segmentation modifications require circuitry equivalent to approximately 7000 logic gates per processor. The size reduction that has been obtained through use of the integrated circuits allows this to be placed in one 19-inch rack with vertical back-plane wiring. It is estimated that implementation of equivalent logic with DEC Flip-Chip cards would require at least three racks.

All fabricated parts, printed-circuit boards, and wired bins are due to be delivered during September, 1966. Debugging should commence by October 1, 1966.

REFERENCES

*Work done under the auspices of the U. S. Atomic Energy Commission.

[1] F. J. Corbato and V. A. Vyssotsky, "Introduction and Overview of the Multics System." Proceedings of the Fall Joint Computer Conference, 1965 (Spartan Books, Washington, D. C., 1965), p. 190.

Table I. Segments for the four page sizes.

Page size	Maximum no. of pages/segment	Maximum segment length
2^7 = 128	2^5 = 32	2^{12} = 4K
2^9 = 512	2^5 = 32	2^{14} = 16K
2^{11} = 2048	2^4 = 16	2^{15} = 32K
2^{13} = 8192	2^2 = 4	2^{15} = 32K

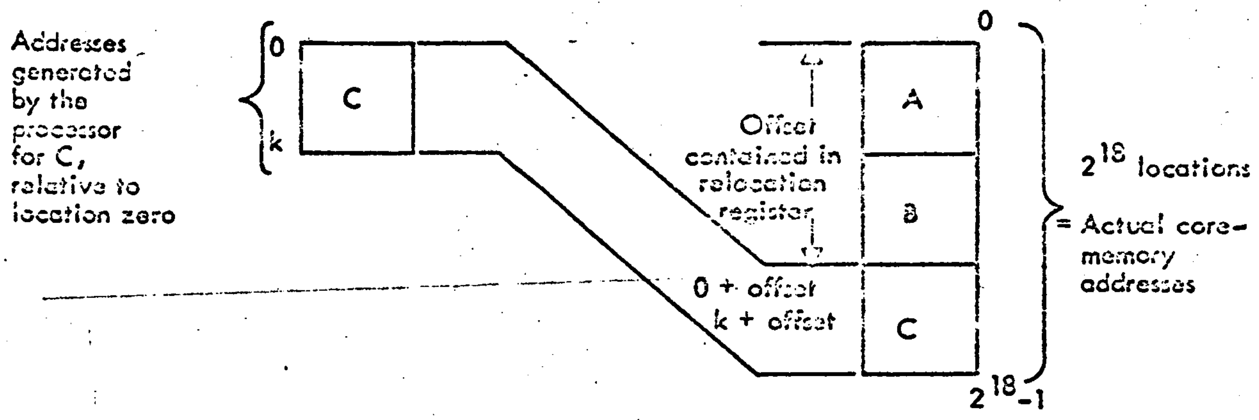

Addresses generated by the processor for C, relative to location zero
0
k
C
Offset contained in relocation register
0 + offset
k + offset
0
A
B
C
$2^{18}-1$
2^{18} locations
= Actual core-memory addresses

Fig. 1. Simple address relocation.

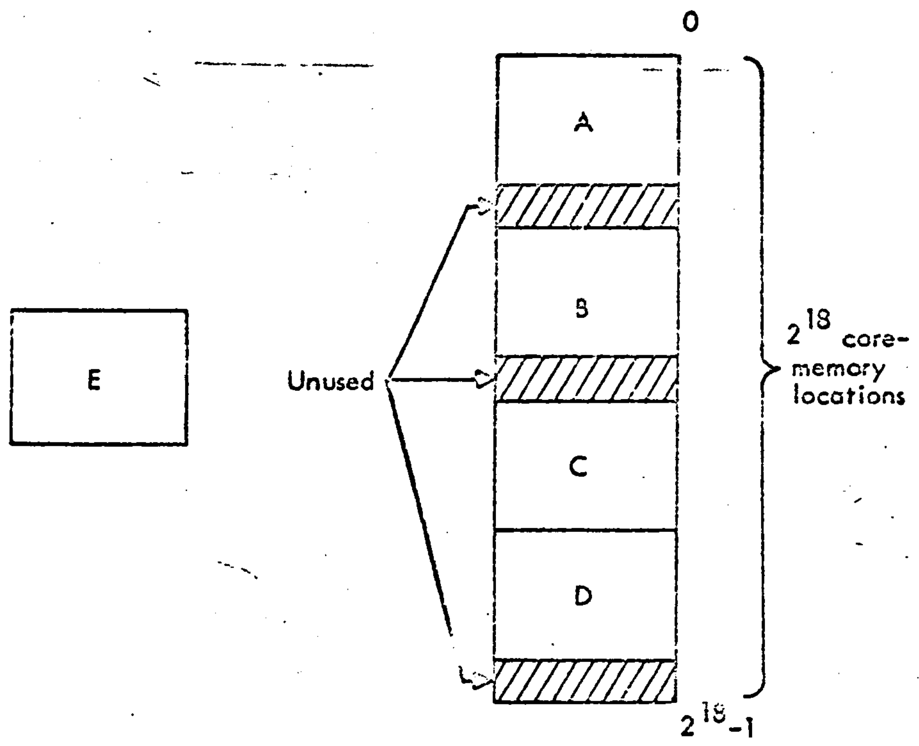


Fig. 2. The problem of core management with simple relocation.

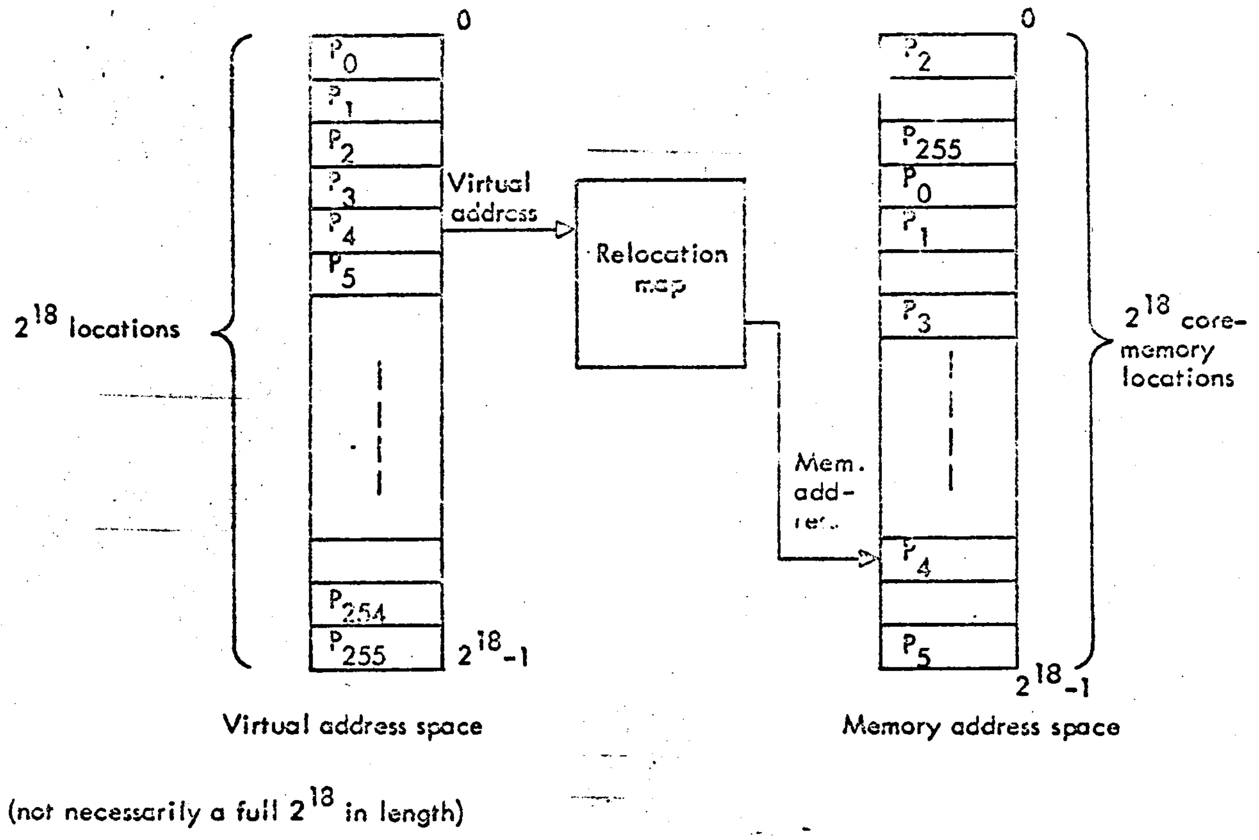


Fig. 3. Relocation by Pagination.

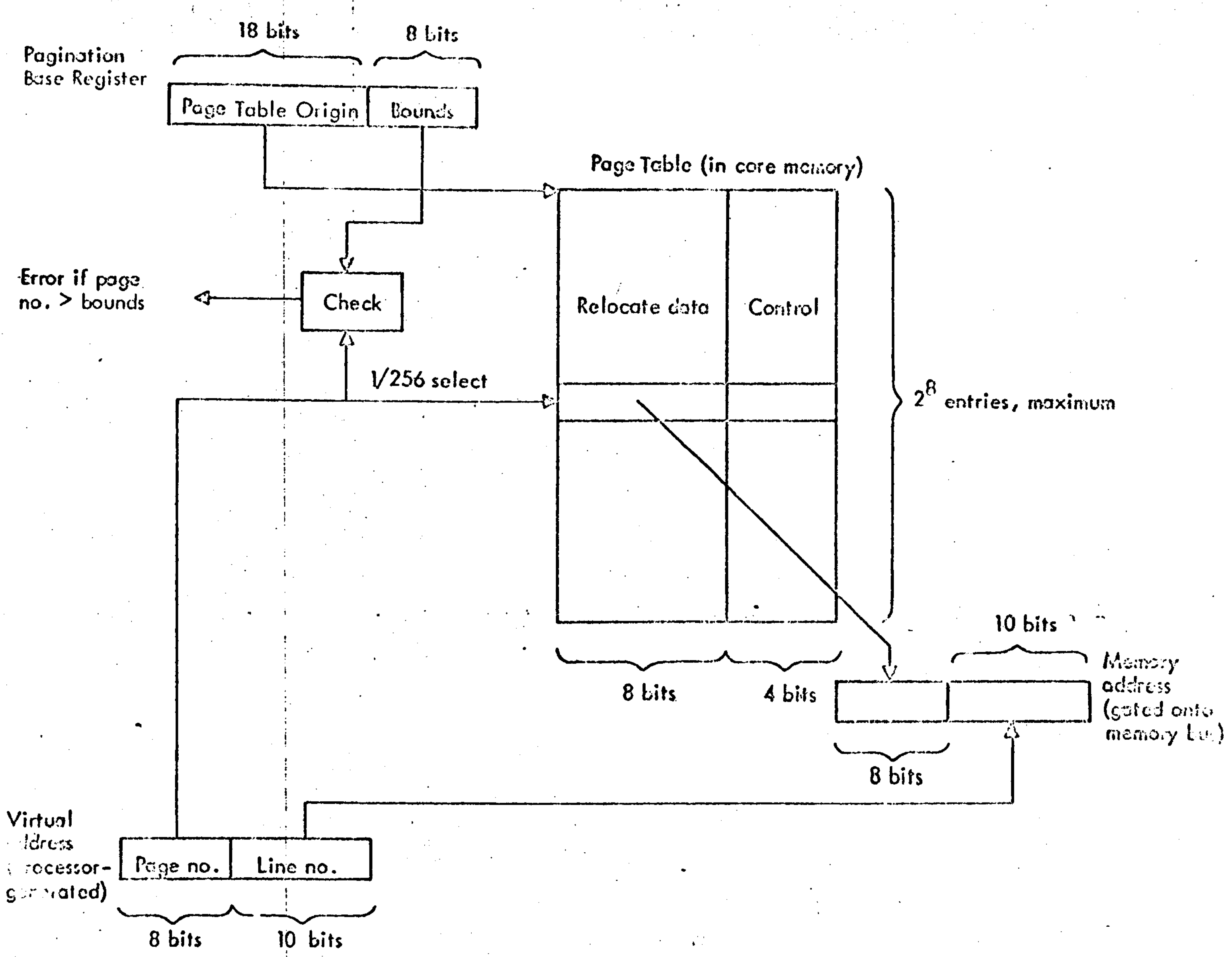


Fig. 4. Implementation of pagination.

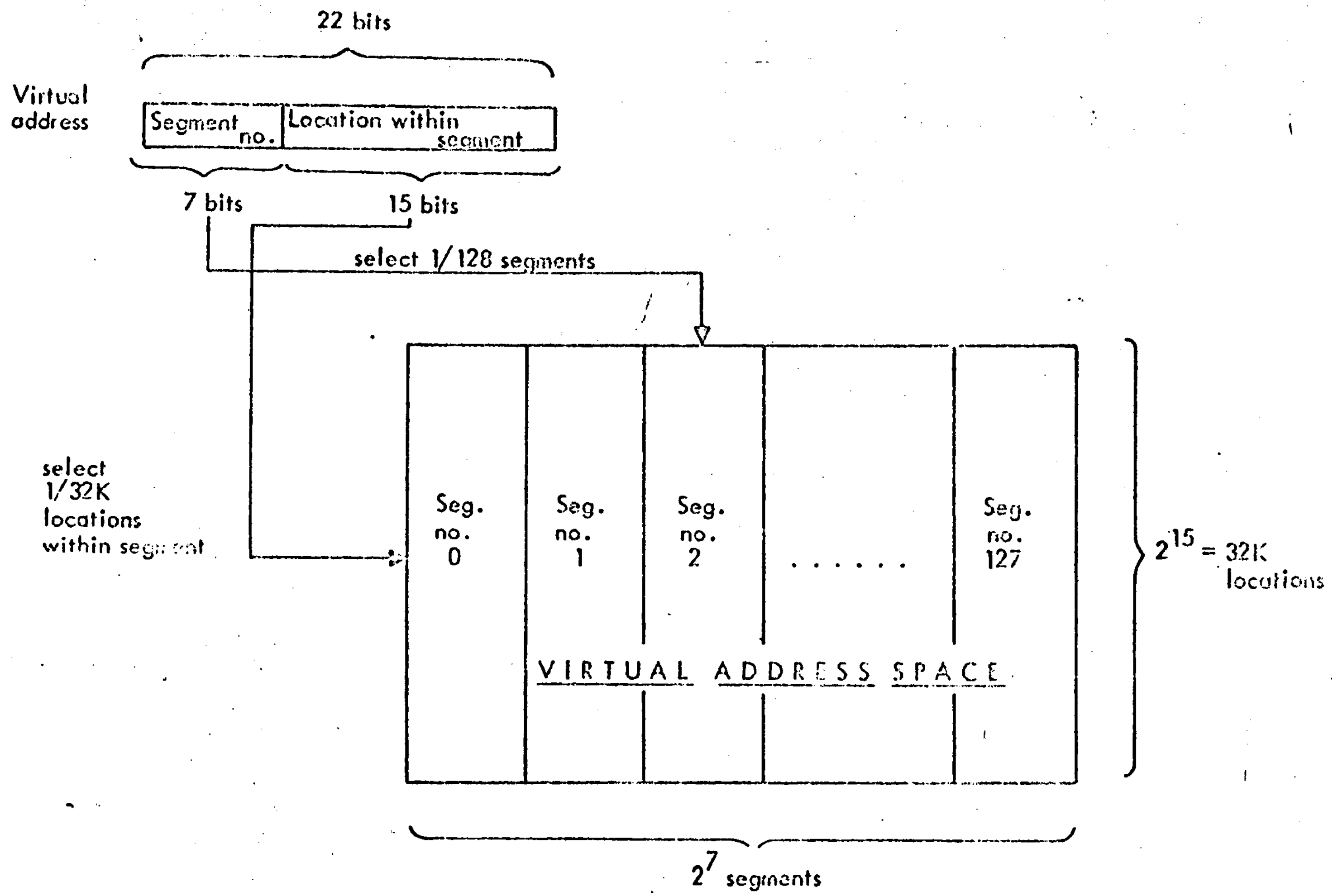


Fig. 5. Two-dimensional virtual address space with segmentation.

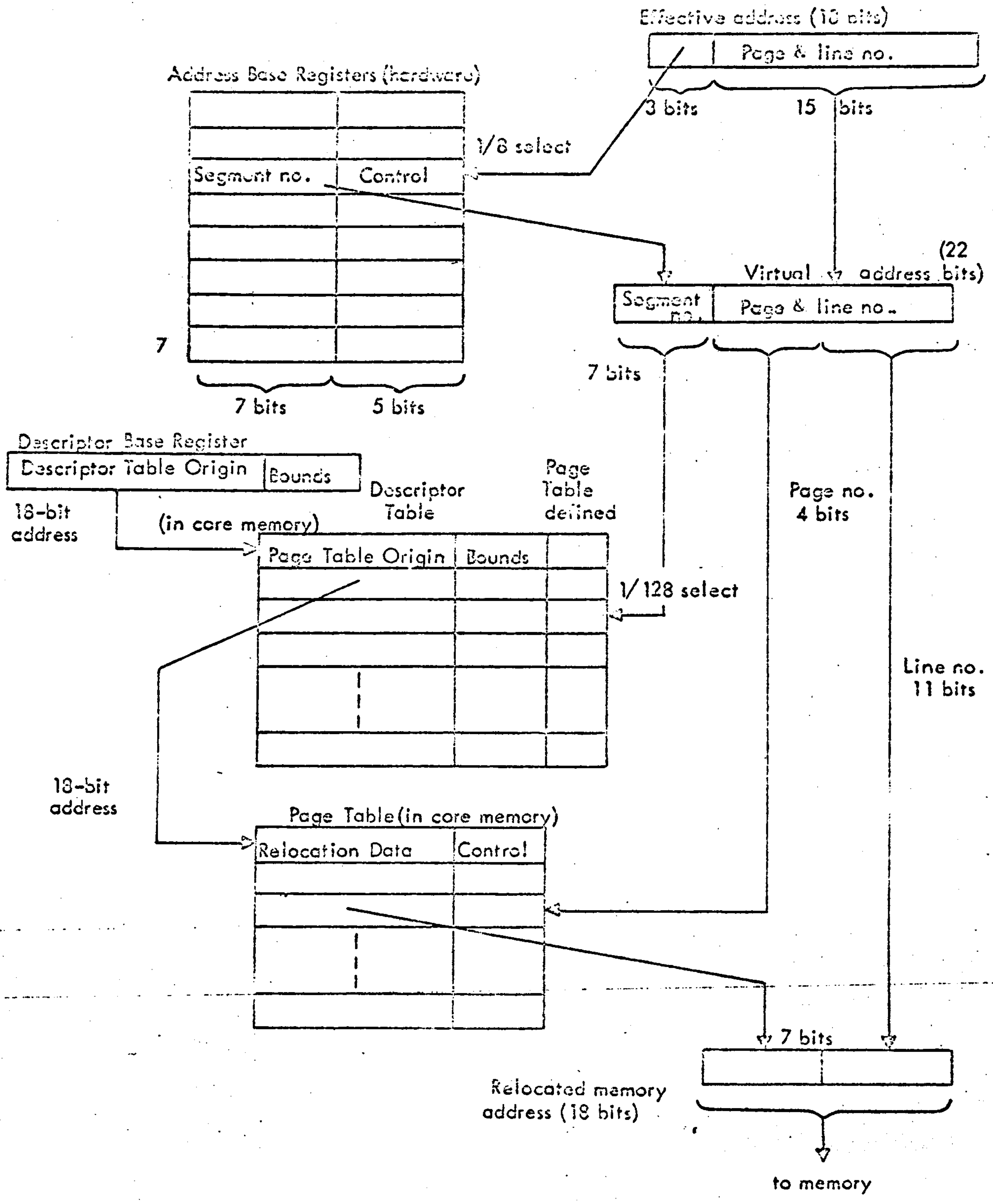


Fig. 6. Basic segmentation with pagination — 2048-word pages.

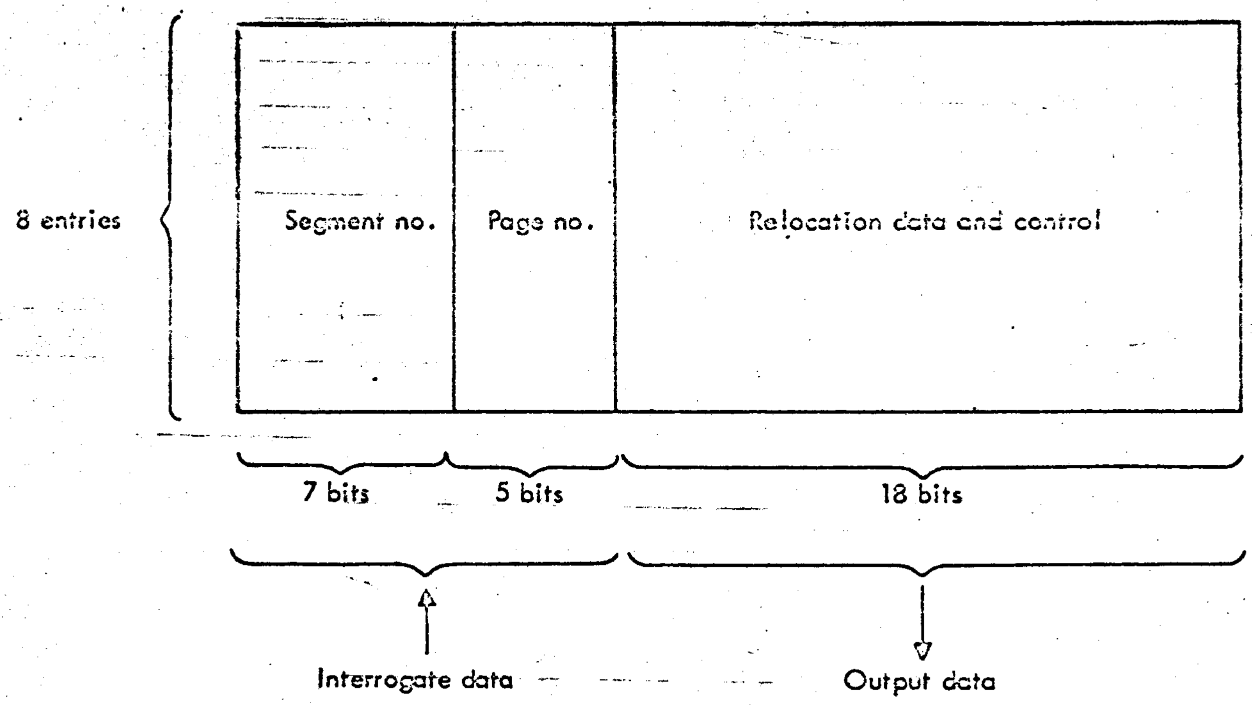


Fig. 7. Associative memory.

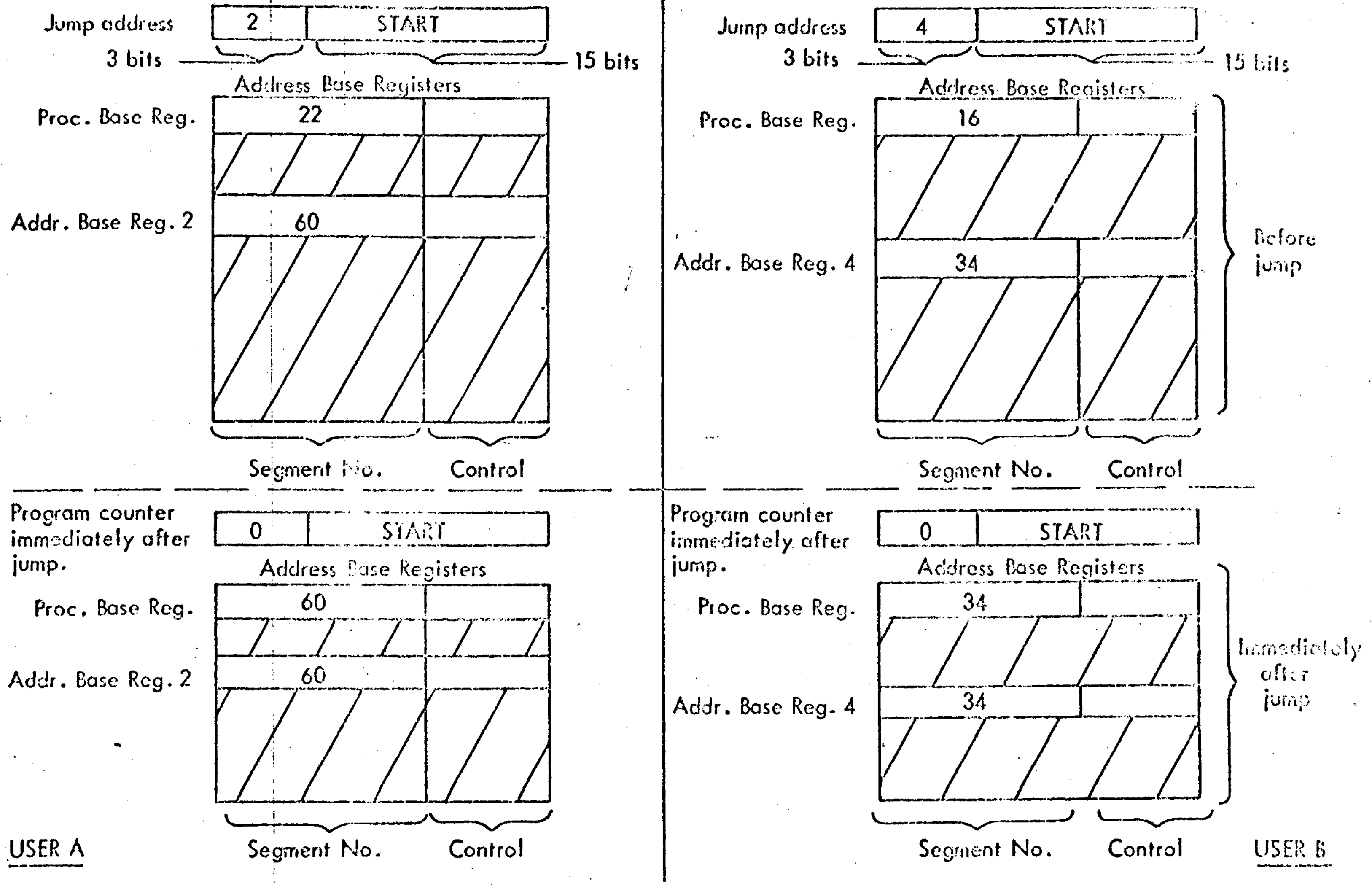


Fig. 8. Example of the assignment of different segment numbers to a common subroutine by two users.

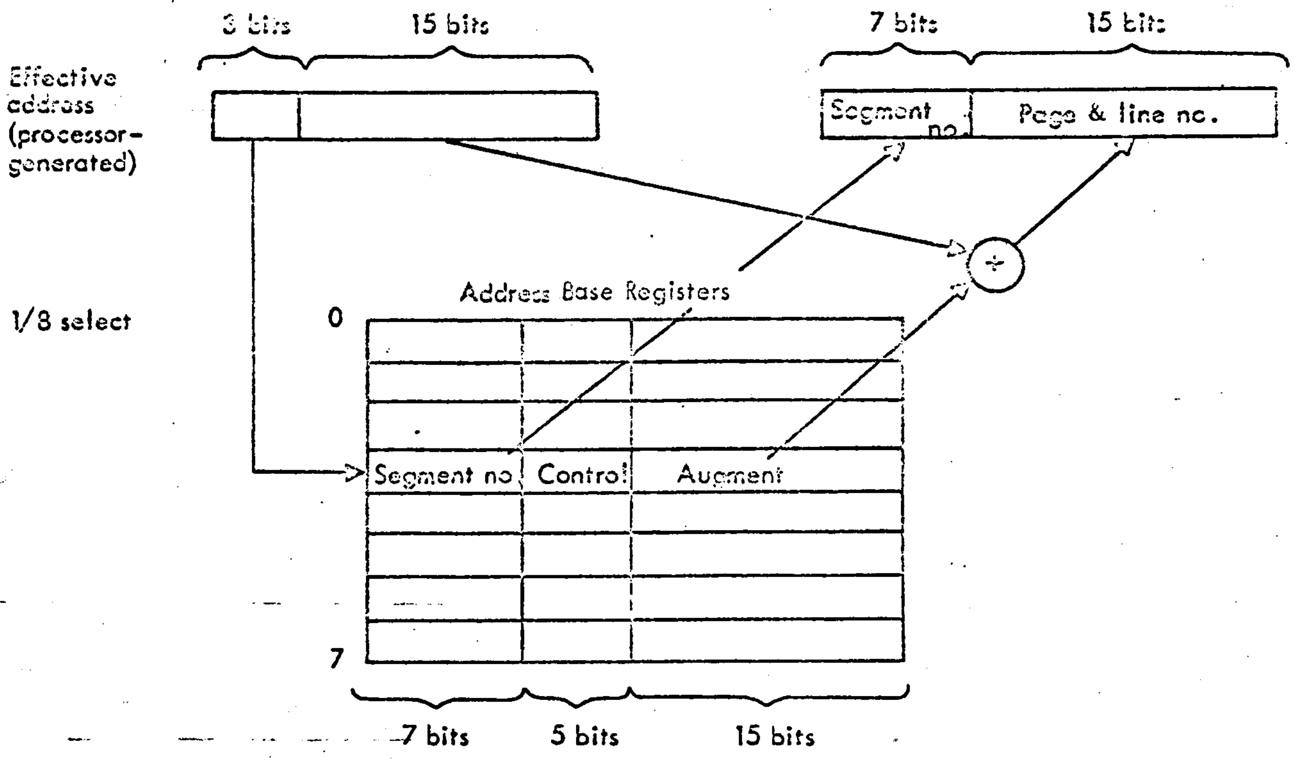


Fig. 9. Generation of virtual address with Augmentation.

1.0 INTRODUCTION

The PDP-6 Segmenting & Paging System consists of hardware modifications to the PDP-6 Central Processors, Type 166, to facilitate reliable and efficient time share operation in the OCTOPUS system. These modifications are visible to system programmers primarily as changes in the address relocation mechanisms.

A two-dimensional addressing system has been incorporated with addressing capability of 2^7 in one dimension and a maximum of 2^{15} in the other, providing a potential address space of 2^{22} locations. A given process consists of one or more segments (to a maximum of 2^7) where each segment is individually paged (with a maximum of 2^{15} locations per segment).

A segment consists of an ordered set of information of sufficient distinction to be isolated and given a name. Any segment may grow or shrink during execution with a maximum size of 2^{15} locations. All addressing within a segment is with respect to location zero. References to other segments are made indirectly through hardware registers (Address Base Registers). Therefore, the largest piece of code that must be loaded as an entity is a segment.

Segments also allow the efficient implementation of reentrant codes by treating them as pure procedure (execute only) segments. A single copy of a pure procedure may be shared by many users, each with its own data segment containing the data unique to the corresponding user. This is particularly useful for large commonly used codes such as compilers and assemblers since considerable core savings can result. Pure procedure segments also insure the integrity of instruction sequences since the segment may not be referenced as data and therefore modified. Consequently it is expected that most system programs will be written in user mode where the protection of pure procedure segments is available.

Paging is a separate feature from segmenting. It permits a correspondence between the effective address as computed by the code and the actual physical core location so that noncontiguous core blocks appear contiguous to the user. Constraints may be placed individually on the pages for protection. Also, paging allows efficient implementation of the concept of single-level storage providing a large address space for the user (exceeding core capacity), and efficient core usage through page turning (only the currently most active pages of a particular process resident in core at a given instant in time).

The remainder of this System Description will emphasize the characteristics of the PDP-6 Segmenting and Paging System. Efficient and sophisticated system programs require an appreciation for both the strengths and weaknesses of the system as implemented.

2.0 DEFINITIONS

SEGMENT	An ordered set of information of sufficient importance or distinction to be isolated and given a name.
COMPUTATION	A set of segments which comprise a meaningful job.
PROCEDURE SEGMENT	A segment which consists primarily of instructions.
DATA SEGMENT	A segment which consists primarily of data.
PAGE	A fixed block of contiguous addresses within a segment. A page is the smallest block of address space which may be uniquely relocated.
EFFECTIVE ADDRESS	An address which may have been indexed and/or indirected prior to relocation.

APPROVED	LAWRENCE RADIATION LABORATORY	ELECTRONICS ENGINEERING	UNIVERSITY OF CALIFORNIA	[illegible] SYSTEMS [illegible] No. 2 Pt. VI
CHECKED		LIVERMORE, CALIFORNIA		
[illegible] D. Pehrson [illegible]	PDP-6 SEGMENTING AND PAGING SYSTEM Revised System Description With Programming Notes			[illegible] 7/7/66 [illegible] 2/68

3.0 MODES OF OPERATION

The two existing modes of operation of the PDP-6, Executive Mode and User Mode, are retained. A DEC modification has been installed which allows I/O operations in User Mode under certain conditions.

All addressing is absolute in Executive Mode. No checks of any kind are made on the effective address which is also the absolute address. Privileged instructions may always be executed. This class of instructions includes I/O operations and access to some of the hardware registers added for segmenting and paging. Interrupts in User Mode result in a change of machine state to Executive Mode if a JSR instruction is given at the interrupt location 4Ø + 2j, or in location 41 + 2j as a result of a BLKI or BLKO overflow.

Addressing in User Mode is always relocated through use of the segmenting and paging features. Checks for bounds violations or illegal use are always made. I/O instructions are allowed in User Mode when the CPA IOT USER flip-flop has been set by Executive Mode programs. Access to the restricted segmenting and paging registers is <u>never</u> allowed in User Mode.

Central Processor sequencing has been modified to allow execution of I/O operations at location 41_8 after trapping to location $4Ø_8$ from either User or Executive mode. This I/O execution is never trapped, whether or not CPA IOT USER has been set.

Second Edition

APPROVED	LAWRENCE RADIATION LABORATORY	ELECTRONICS ENGINEERING LIVERMORE, CALIFORNIA	UNIVERSITY OF CALIFORNIA	Systems Report No. 2 Pt. XI
CHECKED				
ENG. D. Johnson	PDP-6 SEGMENTING AND PAGING SYSTEM Revised System Description With Programming Notes			7/7/66
DRAWN CD				PAGE 8 OF 34 10/66

4.0 <u>SYSTEM DEFINITION</u>

4.1 <u>General</u>

Memory addressing in Executive Mode is not modified from that of the PDP-6 System as described by D.E.C. All references to segmenting, or paging features imply execution in User Mode.

An 18-bit address is relocated by breaking it down into three parts. This is shown in Figure 1 for a page length of 2^{11}= 2048 words.

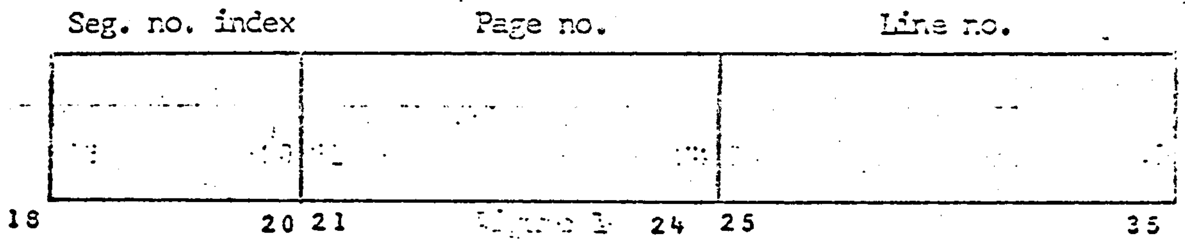


FIGURE 1

The most significant three bits define a segment number after an indexing process. Some lower order bits (four bits for a page length of 2^{11} words) define a page number within the segment while the remaining less significant bits define a line number within the page. A page is the smallest amount of address space which may be arbitrarily relocated.

A computation may contain a maximum of 2^7-1 = 127 segments (numbered $Ø_8$ through 176_8). Segment number 177_8 is illegal and will cause an interrupt to Executive Mode.

Four page sizes are defined, with the constraint that all pages in a segment must be the same length.

2^7	=	128 words/page
2^9	=	512 words/page
2^{11}	=	2048 words/page
2^{13}	=	8192 words/page

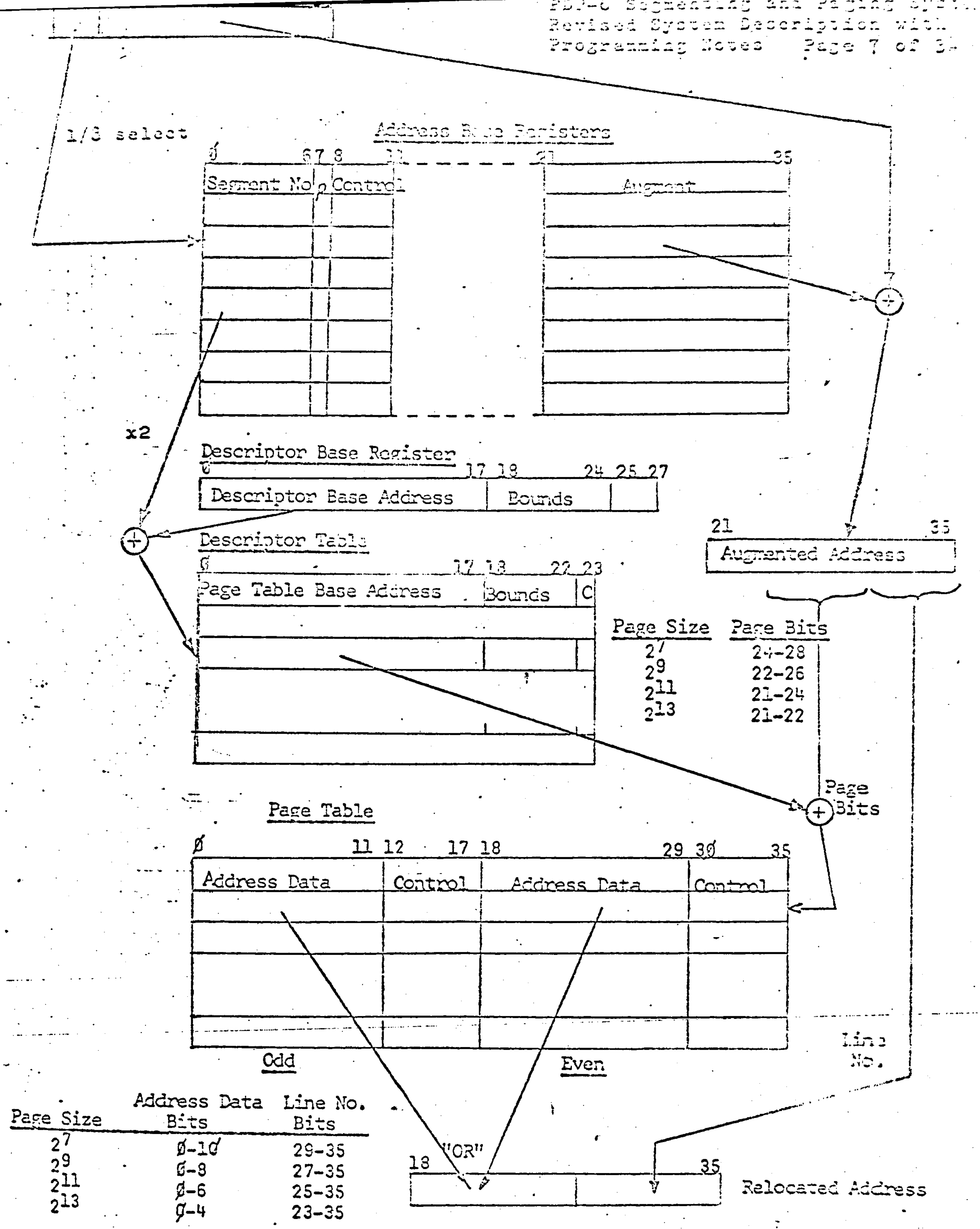
A segment can contain up to 32 pages or 2^{15} locations, whichever constraint is met first. Hence, the maximum number of pages and maximum segment length for the four page sizes are:

Page Size	Max. No. of Pages	Max. Segment Length
$2^7 = 128$	$2^5 = 32$	$2^{12} = 4096$
$2^9 = 512$	$2^5 = 32$	$2^{14} = 16384$
$2^{11} = 2048$	$2^4 = 16$	$2^{15} = 32768$
$2^{13} = 8192$	$2^2 = 4$	$2^{15} = 32768$

The effective address space available to any user process in the modified system has therefore been increased from 2^{18} locations in the unmodified system to a maximum of 2^{22} locations (2^7 segments x 2^{15} locations per segment, for 2^{11} or 2^{13} page sizes).

An address (contained in the MA of the PDP-6) is relocated by expanding it first to a 22-bit address, adding a 15-bit second index (called an Augment) and then relocating the resultant via the paging mechanism. This relocation process is outlined in Figure 2.

The significant three address bits (MA 18-20) select one of the eight Address Base Registers which provide a seven-bit Segment Number. The remaining 15-address bits (MA 21-35) are added to the Augment field of the selected Address Base Register. The 22-bit address formed by concatenating the seven-bit Segment Number with the resultant 15-bit Augmented Address is then relocated via the paging hardware.



Page Size	Page Bits
2^7	24-28
2^9	22-26
2^{11}	21-24
2^{13}	21-22

Page Size	Address Data Bits	Line No. Bits
2^7	Ø-10	29-35
2^9	Ø-8	27-35
2^{11}	Ø-6	25-35
2^{13}	Ø-4	23-35

Address Relocation
Figure I

A Page Table exists in core for each segment of a process which runs in User Mode. One Descriptor Table in core is associated with each computation with entries for each segment. Descriptor Tables have two words per segment so are 2N words long for a process containing N segments. Each page entry in the Page Table requires 18 bits so that the Page Table is $m/2$ words long for a segment consisting of m pages (m even or rounded to next higher, even, integer). This saves core memory since each Page Table word contains relocation information for two pages.

The segment number in the selected Address Base Register, multiplied by two, provides an index into the Descriptor Table whose origin is given by the Descriptor Base Address contained in the Descriptor Base Register. The Descriptor Table entry gives the Page Table Base Address which is the origin of the Page Table for that particular segment.

The desired page within the segment is given by the more significant bits of the Augmented Address adjacent to those bits defining the selected line number. These bits are:

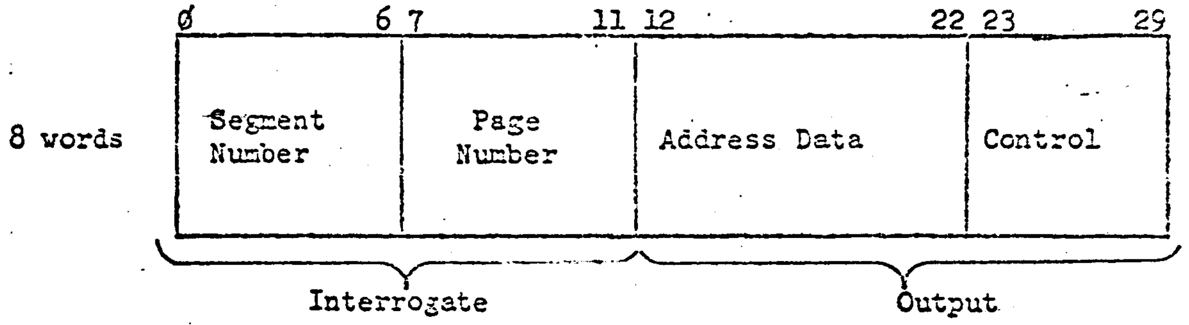
Page Size	Augment Address Bits Defining Page Number
2^7	Bits 24 - 28
2^9	Bits 22 - 26
2^{11}	Bits 21 - 24
2^{13}	Bits 21 - 22

Where the page size is defined by bits 8 and 9 of the selected Address Base Register. This points to a unique half-word entry in the selected Page Table whose Address Data is substituted for [illegible] and page number information producing the relocated address. Many checks are made throughout the relocation process which are described in subsequent sections.

4.2 Operation With an Associative Memory

An overhead of two memory cycles (one to the Descriptor Table and one to the Page Table) is required for the relocation of an address using the mechanism just described. This 200% increase in time is intolerable if required for relocation on every memory reference.

An Associative Memory has therefore been provided to store the required relocation information for the eight most currently accessed pages. Its organization is shown in Figure 3.



Associative Memory

Figure 3

The seven-bit segment number obtained from Address Base Register and five-bit page number obtained from the Augmented Address are simultaneously matched against the corresponding information in the eight words of the Associative Memory. If a match is obtained, the Address Data and Control information in the right half of the word with the match are gated out. This data is identical to the corresponding information in the Page Table accessed as described in Section 4.1.

When no match is obtained, memory cycles to access the Segment and Page Tables must be taken. However, the resultant data from the Page Table and the segment and page numbers are written into the Associative Memory replacing the word referenced the longest ago. The implied strategy assumes that the most recent history of page usage is the best predictor of future page usage and has the advantage of not being too difficult to implement at a hardware level.

Complex programs which reference large portions of core in random patterns should therefore use large page sizes since the amount of core mapped by the Associative Memory at any instant in time is a linear function of page size. A maximum of 64k (8 pages x 8k page size) may be mapped without causing extra memory references to the paging tables. The efficiency in paging gained by the use of large page sizes is counter-balanced by the additional constraints placed on the core mapping algorithms so that large pages should be used only with discretion.

The real-time required for address relocation varies. Addresses which use Address Base Register - Ø and are augmented by zero require the same time for relocation as required on the original PDP-6 for address relocation through the addition of a constant. An additional 1ØØ nanoseconds are required for addresses referencing any other Address Base Register to allow the augment to be added prior to relocation. Executive Mode addressing operates as formerly.

4.3 Registers

Twenty-one new registers have been added to each processor (including eight registers comprising the Associative Memory) to implement segmenting and paging. The eight-bit Relocation and Protection Registers in the unmodified processor will be removed. The DATAO instruction to Device Number ØØØ (the processor) which provided access to the Relocation and Protection Registers now has no effect.

The added registers are:

a. Eight Address Base Registers where Address Base Register - Ø is also called the Procedure Base Register.

b. One Descriptor Base Register.

c. Two Interrupt Summary Registers.

d. Eight registers comprising the Associative Memory.

e. One Data Buffer Register (not visible to programmer).

f. One Adder Buffer Register (not visible to programmer).

The following access exists in User Mode:

I. Read Access

a. Address Base Register (including the Procedure Base Register).

b. Associative Memory.

c. Descriptor Base Register.

II. Write access to the Address Base Registers only (except the Procedure Base Register, Address Base Register 0).

In Executive Mode, the following access exists:

I. Read Access

a. Address Base Registers (including the Procedure Base Register).

b. Associative Memory.

c. Descriptor Base Register.

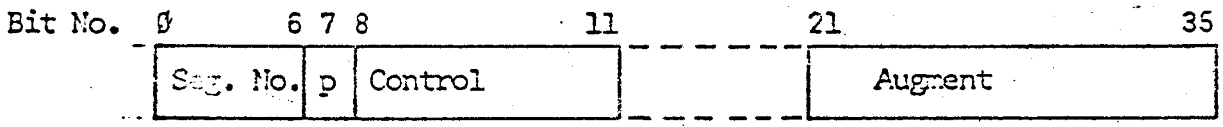
d. Interrupt Summary Registers.

II. Write Access

a. Address Base Registers (including the Procedure Base Register).

b. Descriptor Base Register.

4.3.1 Address Base Register i (i = Ø-7)



Bits Ø-6: Segment number

Bit 7 Odd parity over bits Ø-6

Bits 8 & 9: Page size definition

Bit No. 8	9	Page Size
Ø	Ø	128 word length
Ø	1	512 word length
1	Ø	2Ø48 word length
1	1	8192 word length

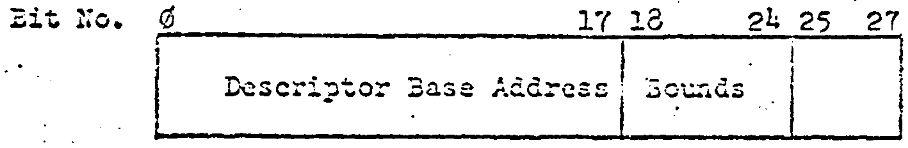
Bit 1Ø:	Set to "1" when register is loaded.
Bit 11:	Unused.
Bits 12-2Ø:	Nonexistent.
Bits 21-35:	Augment index.

Odd parity (bit-7) over the segment number must be software generated prior to loading the register with the Load Register (LDR) command (see Section 4.5.1). The LDR command loads only bits Ø-7 and bits 21-35 of the register. Bit-1Ø is set to "1" by hardware and the page size bits (8&9) are set by hardware by reference to the Descriptor Table (see Section 5.1).

The Augment field of the Procedure Base Register will always contain zero so that the effective address is indexed by zero when the Procedure Base Register is used to save time. The Augment field is forced to zero by hardware when the Procedure Base Register is loaded with an LDR in Executive Mode. It is also forced to zero when information is gated into the Procedure Base Register.

Bits Ø - 6 of all Address Base Registers will be set to all 1's when the Descriptor Base Register is loaded. Since Segment number 177_8 is illegal, reference to an Address Base Register not loaded for, or by, a user will be trapped.

4.3.2 Descriptor Base Register



Bits ∅-17: Origin (absolute) of the Descriptor Table.

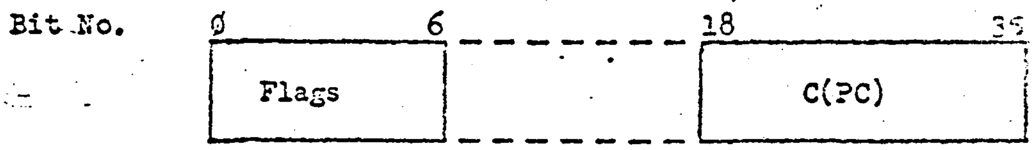
Bits 18-24: Number of the largest valid segment entry in the Descriptor Table.

Bits 25-27: Unused.

Bits ∅-24 of a memory word are loaded into the Descriptor Base Register with an LDR command. The remaining bits have no effect.

A bounds violation exists if the segment number is strictly greater than the Bounds contained in bits 18-24.

4.3.3 Interrupt Summary Register-1



This information is sampled early in the cycle (FT5) of each instruction executed in User Mode. If PI SYNC=1 or the processor is in Executive Mode, the last sampled contents of this register are retained.

Bits ∅-6: Flags with the following definitions:

Bit No.	Definitions
∅	AR OV FLAG
1	AR CRY∅ FLAG
2	AR CRY1 FLAG
3	AR DC CHG FLAG
4	CHF7 (BIS)
5	EX USER
6	CPA IOT USER

Bits 18-35: Contents of the PC before incrementing and unrelocated.

Therefore, a program may be restarted at the location at which difficulty occurred if not of the catastrophic type. The following instruction set sequence will return to the interrupted location for interrupts due to the Segmenting and Paging System

			Comments
STR	13,	LOC	LOC = C(Int Sum Reg-1)
JRST*	11,	LOC	Indirect return to User Mode to C(PC) = C(Int Sum Reg-1) right

Note: See Section 4.5.2 for definition of the STR order.

Normal I/O interrupts should be handled as at present.

4.3.4 Interrupt Summary Register-2

Bit No.	0 — 15	18 — 35
	Indicators	Address

This information is set when conditions are detected by the segmenting and paging system which should cause an interrupt to Executive Mode. The data in this register provide information for diagnosis of the cause of the interrupt while the data in Interrupt Summary Register-1 preserve program continuity.

Bits 0-15: Indicators with the following definitions:

	Bit No.	Definition if set to 1
Address Errors	0	Parity failure on segment number
	1	Descriptor Table bounds violation
	2	Page Table bounds violation
	3	Segment undefined (177_8)
	4	Segment overflow
Information not in core	5	Segment not in core
	6	Page not in core
Illegal Page Usage	7	Procedure bit not enabled
	8	Data read bit not enabled
	9	Data write bit not enabled
	10	Illegal instruction in User Mode
Time of Failure	11	Failure at Instruction time
	12	Failure at Address time
	13	Failure at operand Fetch time
	14	Failure at operand Store time
	15	Failure on I/O memory reference
	16	Paging Hardware error

Reference Section 5.7 for a detailed description of the conditions which set these bits.

Bits 18 - 35: Address, augmented, which could not be successfully relocated causing the interrupt. This address can be interpreted through analysis of bits 0-15.

Read access, STR, to both Interrupt Summary Registers is allowed only in Executive Mode. Load access does not exist since these registers are always set by hardware.

4.3.5 Associative Memory Registers

Bit No.	∅ – 6	7 – 11	12 – 22	23	24 – 29
	Seg. No.	Page No.	Address Data		Control

Bits ∅- 6:	Segment Number (no parity corresponding to data in bits 12-29.
Bits 7-11:	Page Number corresponding to data in bits 12-29.
Bits 12-22:	Address data which is substituted for the high order bits of the effective address for purposes of relocation.
Bit 23:	Unused.
Bit 24:	If 1, the page is in core and may be accessed.
Bit 25:	If 1, the page may be executed as procedure.
Bit 26:	If 1, the page may be read as data.
Bit 27:	If 1, data may be written into the page.
Bit 28:	If 1, the page has been written into (and therefore must be copied onto disc if it is desired to remove it from core). This bit is set to 1 along with the corresponding bit in the Page Table if a memory store operation takes place and this bit is ∅ in the Associative Memory.

Bit 29: Set to 1 (in the Associative Memory and Page Table) when the entry is brought into the Associative Memory by segmenting and paging hardware.

Three-bit counters associated with each Associative Memory register keep track of relative page use so that the page referenced longest ago is uniquely determined.

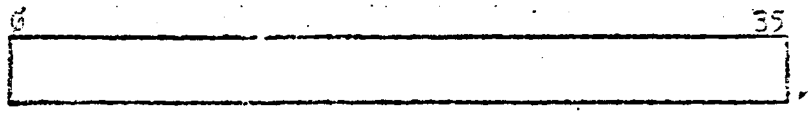
Bit 29 will always be 1 for all entries.

Bit 28 may be Ø for a page with read-only access. Then, the page need not be copied back onto disc when no longer required in core.

Bits Ø-6 are set to all 1's (177_8) in all Associative Memory registers when the Descriptor Base Register is loaded. Since segment number 177_8 is not allowed, a match will never result. This is used to preset the Associative Memory to prevent an erroneous match when initiating a new active user.

4.3.6 Data Buffer Register

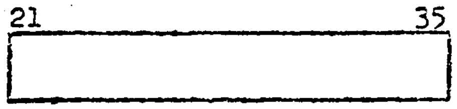
Bit No.



This hardware register is not program accessible. It is used for buffering C(MB) when accessing the Segment and Page Tables to bring an entry into the Associative Memory i.e., it holds the information obtained from the Descriptor Table while referencing the Page Table. It also buffers data on the added segmenting and paging instructions (see Section 4.5) to eliminate hardware fan out difficulties on the MB.

4.3.7 Adder Buffer Register

Bit No.



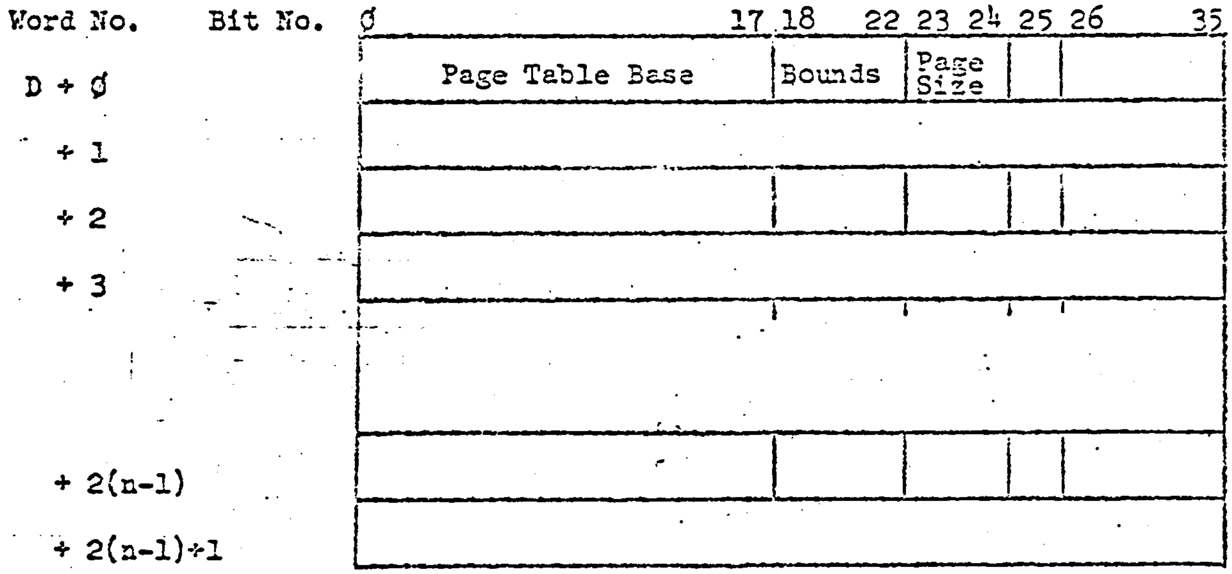
This register is not program accessible. It is used to buffer the "augmented transfer to address" on JUMP instructions. Its use solves a hardware logical conflict which exists because of the Augment feature which has been incorporated. Reference Section 5.3 on JUMP orders.

4.4 Core Memory Tables

4.4.1 Descriptor Table

The Descriptor Table is a variable length table which contains a two-word entry for each segment which may be addressed by a process in User Mode. The maximum segment number (beginning with Ø) which may be addressed is given by the Bounds field of the Descriptor Base Register (see Section 4.3.2). If the Bounds field = n-1 (for n segments), the Descriptor Table must be 2n words long. If a segment between Ø and n-1 is not pertinent to the process, it may be defined as nonexistent in which case there is no need for a Page Table for that segment.

The Descriptor Table is generated and kept up to date by Executive Mode system programming.



D = Descriptor Base Address (from Descriptor Base Register).

Bits Ø-17:	Origin (absolute) of the corresponding Page Table.
Bits 18-22:	Number of the largest valid page number in the page table.

Bits 23-24: Define page size:

Bit No. 23 24	Page Size
∅ ∅	2^7 = 128 words
∅ 1	2^9 = 512 "
1 ∅	2^{11} = 2∅48 "
1 1	2^{13} = 8192 "

Bit 25: If 1, a Page Table exists for this segment and it is initially assumed (until access to the Page Table) that some pages of the segment are in core. Access to the Page Table is denied and "Segment Not in Core" is set if this bit is ∅.

Bits 26-35: Undefined.

Bits ∅-35 of the odd words are also undefined.

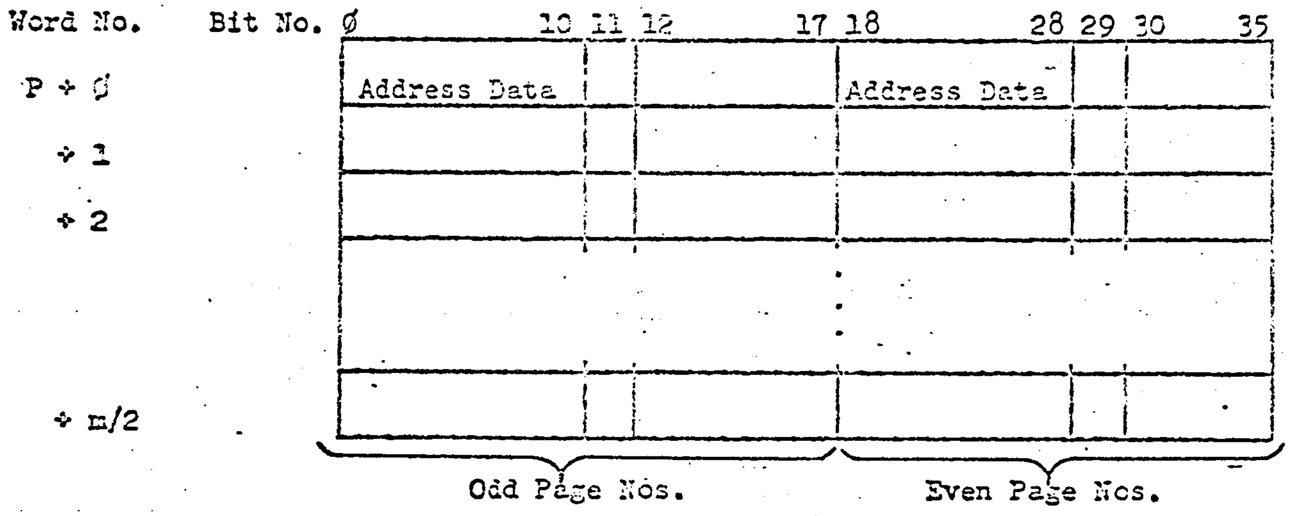
Since bits 26-35 of even words and all odd words are not used by hardware, they are available for use by software for segment related data.

Page size is defined by the Descriptor Table (which in turn is defined by Executive Mode programs). The page size bits in the Address Base Register are set equal to these [illegible] in the Descriptor Table the first time an Address Base Register is referenced (Address Base Register Bit-1∅=1). See Section 4.3.1. This insures that a user cannot gain access to unallowed core space through erroneous definition of a larger-than-intended page size when loading the Address Base Register.

The segment not in core bit denies access to the corresponding Page Table and hence the Page Table need not exist. This gives a core savings for processes whose Descriptor Table is sparsely filled with only a few defined segments (not consecutive).

4.4.2 Page Table

A Page Table exists for each defined segment of a process and consists of $\frac{m}{2}$ PDP-6 core words (where m = number of pages in segment, rounded to larger, even, integer, if odd). Each core word contains relocation data for two pages.



P = Page Table Base Address (from Descriptor Table).

Bits Ø-17 and bits 18-35 are defined as their corresponding bits (positions 12-29) of the Associative Memory. See Section 4.3.5.

Bit 16 or 34 is set to 1 when that page is first written into by the hardware which sets bit 28 in the Associative Memory. Bit 17 or 35, set to 1 when loaded into the Associative Memory, can be used by system routines to keep statistics on page use.

4.5 Additional PDP-6 Instructions

Six new instructions have been added to the PDP-6 instruction set.

4.5.1 Load Register (LDR)

Octal Code: 12Ø

Effect: C(Effective Address) $\longrightarrow$ C(Register)

Where C(Effective Address) are left adjusted with bit locations as defined in the destination register (Section 4.3). The Associative Memory registers are loaded only by hardware sequencing (see Section 5.2). The destination register is defined by the accumulator field of the order.

	ACCUMULATOR FIELD			
Register	(AC8) Bit 8	(AC4) Bit 9	(AC2) Bit 1Ø	(AC1) Bit 11
Address Base Reg-Ø	Ø	Ø	Ø	Ø
Address Base Reg-1	Ø	Ø	Ø	1
•		•		
•		•		
•		•		
Address Base Reg-7	Ø	1	1	1
Unused	1	Ø	Ø	Ø
•		•		
•		•		
•		•		
Unused	1	1	1	Ø
Descriptor Base Reg	1	1	1	1

In User Mode, access to Address Base Reg-Ø (the Procedure Base Register) and the Descriptor Base Reg is not allowed and will trap to Executive Mode. Access is valid in Executive Mode.

Loading the Descriptor Base Reg will set the segment number fields in all Address Base Registers and all Associative Memory entries to all 1's (segment no. 177_8).

4.5.2 Store Register (STR)

Octal Code: 121

Effect: C(Register) ⟶ C(Effective Address)

This is the inverse of the LDR operation. Access to all registers except the Interrupt Summary Registers is allowed in User Mode.

	ACCUMULATOR FIELD			
Register	(AC8) Bit 9	(AC4) Bit 10	(AC2) Bit 11	(AC1) Bit 12
Address Base Reg-Ø	Ø	Ø	Ø	Ø
Address Base Reg-1	Ø	Ø	Ø	1
•		•		
•		•		
•		•		
Address Base Reg-7	Ø	1	1	1
Unused	1	Ø	Ø	Ø
•		•		
•		•		
•		•		
Unused	1	1	Ø	Ø
Int. Summary Reg-1	1	1	Ø	1
Int. Summary Reg-2	1	1	1	Ø
Descriptor Base Reg	1	1	1	1

4.5.3 Store Associative Memory (STAM)

Octal Code: 122

Effect: C(Associative Memory Register) ⟶ C(Effective Address)

where C(Effective Address) are left adjusted.

This instruction is valid in _either_ Executive or User Mode.

The Associative Memory Register (Ø-7) to be read is specified by bits 1Ø-12 of the Accumulator Field of the instruction. Bit 9 is zero.

UNLAM with bit 9 = 1 is executed as a NO-OP.

4.5.4 Relocate Address (RELA)

Octal Code: 123

Effect: $\text{C(AC)}_{\text{right half, relocated}} \longrightarrow \text{C(Effective Address)}_{\text{right half}}$.

This instruction is valid in either User or Executive Mode. If in User Mode, the Effective Address is also relocated.

Reference Section 5.6 for results when relocation of the C(AC) fails.

4.5.5 PUSHJX

Octal Code: 13Ø

Effect: AC+1ØØØØØ1 ⟶ AC

Misc Bits, $\text{C(Proc Base Reg)}_{\text{Ø-7}}$, C(PC) ⟶ C(C(ACRT))

E ⟶ PC (augmented)

Interrupt on overflow

This instruction is similar to PUSHJ with the exception that the segment number contained in the Procedure Base Register is also stored in the pushdown list, along with the miscellaneous flags and the contents of the Program Counter.

```
Bit No.   Ø          6 7    9 1Ø       17 18                 35
          | FLAGS      |      |  SEG. NO  |      C(PC)          |

                         C(C(ACRT))
```

Bits Ø- 6: Miscellaneous bits (flags).

Bits 7- 9: Unused.

Bits 1Ø-17: Segment Number, including parity, contained in the Procedure Base Register.

Bits 18-35: Contents of the Program Counter (after increment).

4.5.6 POPJX

Octal Code: 131

Effect: $C(C(ACRT)) \longrightarrow$ Proc. Base Reg. $_{\text{Ø-7}}$, PC AC-1ØØØØØ1→AC

Interrupt on underflow

This instruction is similar to POPJ with the exception that the segment number bits of the Procedure Base Register are replaced with the segment number bits (Ø-7) contained in the designated entry in the push down list, C(ACRT).

The PUSHJX-POPJX instruction pair have been added for use as the standard linking mechanism for intersegment transfers. Since the "transferred from" segment number is stored in and retrieved from the pushdown list along with the PC contents, the called segment need not load an Address Base Register with the calling segment number prior to giving the return jump. This is very desirable since the requirement that the original calling segment have its segment number in a Base Register, other than the Procedure Base Register, is eliminated. Placing additional constraints on program conventions is therefore eliminated.

5.0 DETAILED OPERATION

A functional flow chart of the segmenting and paging system is shown in Figure 4.

5.1 First Address Base Register Access

Bit 10 of any Address Base Register is set to 1 when it is loaded, with an LDR order or when the Procedure Base Register is loaded as a result of a PUSHJX order. This will inhibit relocation when the Address Base Register is addressed by the significant three Effective Address bits. The page size is defined by access to the Descriptor Table which sets bits 8 & 9 of the Base Register and resets bit 10.

The Descriptor Table access is required so that page size cannot be set directly by users through load access to the Base Registers. The page size therefore may not be changed erroneously by a user during program execution to gain access to unallowed core locations.

It is possible to get a match in the Associative Memory and still get a FIRST ABR ACCESS indication. The case exists if a PUSHJX is followed by the corresponding POPJX where the original entry in the Associative Memory has not been replaced. The POPJX sets bit 10 of the Procedure Base Register to 1 to produce the FIRST ABR ACCESS indication. This is done since the user does have access to the PUSHJX-POPJX push-down list and therefore would have access to the page size lists, had they also been placed in the push-down list.

Since a match is possible with a FIRST ABR ACCESS indication, the re-locatio~~n~~ data cannot be loaded into the Associative Memory location with a count of 7 on the same sequence. If one did, it is likely that two entries in the Associative Memory would be identical, producing faulty hardware sequences.

5.2 No Associative Memory Match

When no match is made in the interrogation of the Associative Memory and the FIRST ABR ACCESS condition is not detected, hardware sequencing to access the Descriptor and Page Tables takes place. Access to the Page Table is in the form of a Read-pause-Write memory cycle so that the page loaded bit in the Page Table (bit 17 or 35) will be set to 1, indicating that the page has been referenced (since it has been loaded into the Associative Memory after a match failure).

LAWRENCE RADIATION LABORATORY

UNIVERSITY OF CALIFORNIA

[illegible] No. 2 [illegible]

ORIG. D. Pehrson

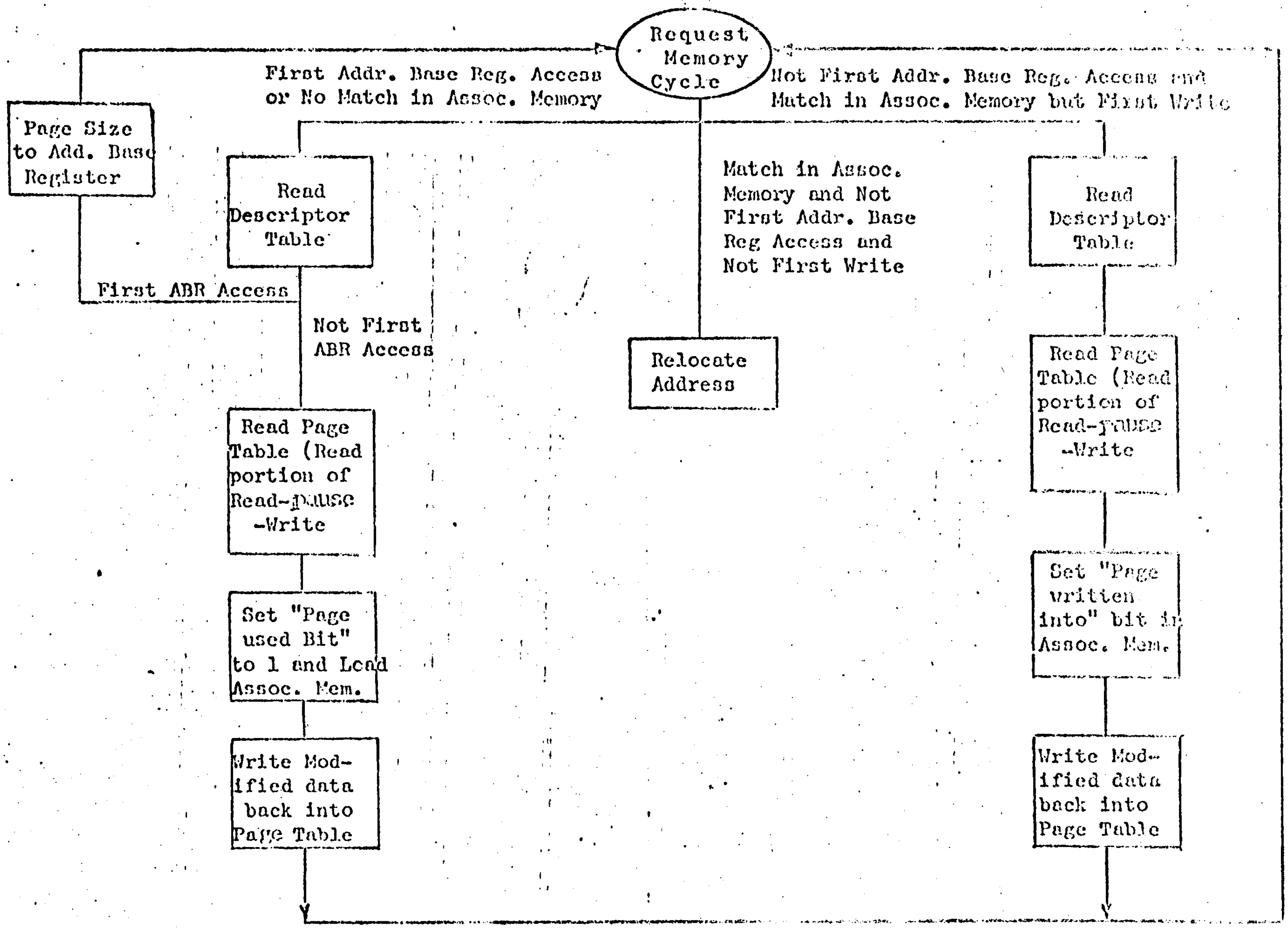
DRAWN [illegible]

PDP-6 SEGMENTING AND PAGING SYSTEM

Revised System Description With Programming Notes.

DATE 7/7/65

PAGE 23 OF [illegible]



Request Memory Cycle
First Addr. Base Reg. Access or No Match in Assoc. Memory
Not First Addr. Base Reg. Access and Match in Assoc. Memory but First Write
Page Size to Add. Base Register
Read Descriptor Table
First ABR Access
Not First ABR Access
Read Page Table (Read portion of Read-pause -Write
Set "Page used Bit" to 1 and Load Assoc. Mem.
Write Modified data back into Page Table
Match in Assoc. Memory and Not First Addr. Base Reg Access and Not First Write
Relocate Address
Read Descriptor Table
Read Page Table (Read portion of Read-pause -Write
Set "Page written into" bit in Assoc. Mem.
Write Modified data back into Page Table

Segmenting & Paging Fun[illegible]al Sequence Chart

Fig.

When the Associative Memory is preset for a new user by loading the Descriptor Base Register, the three bit counters associated with each word are preset to values such that each count from Ø through 7 is uniquely represented once. The first Associative Memory load will be to that word whose count equals 7. After the match takes place, that count is reset to zero and all other counts incremented by one so that all counts Ø through 7 are still uniquely represented. In general, when a match is made, the associated counter is reset to zero and all other counters which contain a count strictly less than the one selected are incremented. This implements the previously defined strategy of retaining the relocation information for the most recently accessed right pages.

Hardware sequencing to the Descriptor and Page Tables also takes place when a successful Associative Memory match results but this is the first time (since the page was brought into core) a write operation has been requested. A Read-pause-Write cycle takes place on Page Table access to mark bit 16 or 34 in the core memory table. Bit 28 of the Associative Memory is also set to 1 so that subsequent write access to the page after the first does not cause hardware sequencing.

5.3 JUMP Instructions

On all JUMP orders, two hardware actions take place when the conditions for jumping are met. Bits 21-35 of the augmented effective address are placed in the PC and bits Ø-11 of the selected Address Base Register are transferred to the Procedure Base Register. [illegible] is transferred to the augment field of the Procedure Base Regist[illegible] and the significant three PC bits. Placing the augmented [illegible] address in the PC eliminates the time required to perform the augment addition when memory is accessed via the Procedure Base Register. This saves approximately 1ØØ nanoseconds in the time required for relocation.

The hardware sequence requires the use of the Adder Buffer Register (Section 4.3.7). The nonaugmented effective address (contents of the MA) are replaced with the augmented effective address prior to the hardware sequence transferring the contents of the MA to the PC.

5.4 Programmed Operators

The 64 instruction codes with zeroes in the first three bits have been reserved as programmed operators. When used, bits Ø-17 of the instruction are stored in bits Ø-17 of location 4Ø. The effective address <u>without</u> augmenting is stored in bits 18-35 of location 4Ø. The instruction at location 41 is then executed.

5.5 Augment Adder

The Augment Adder is an 18-bit parallel binary adder with worst case add time of 75 nanoseconds. This adder is also used for adding the segment number to the Descriptor Base Address and the page number to the Page Table Base Address.

In its normal use, it functions as a 15-bit adder since the Address Base Register is selected by the significant three bits of the effective address. The augment field of the selected Base Register then serves as one input to the adder for the remaining 15-bit intrasegment address.

Hardware checks will detect erroneous augments which generate segment addresses greater than the maximum segment length for the defined page size, rather than allow these to be interpreted by their low-order bits only. An interrupt will result with "Segment Overflow" (bit 4 of Interrupt Summary Register 2) set.

5.6 Interrupts and Program Continuity

The PDP-6 may strobe for an interrupt request at any of the following times during instruction execution:

a.	ITØ -	Just before fetching a new instruction.
b.	ATØ -	Just before indexing an address (at each level of indirection).
c.	BLT T4 -	After each word has been transferred (the incremented AC is stored).
d.	ATØ -	Same as (b), except that the case occurs after incrementing the pointer word for character operations.

Not all interrupts are caused by illegal addressing. Addresses which are not in core, due to either segment or page not in core conditions, produce interrupts which should cause the desired page to be obtained. Program continuity must be retained for resumption at a later time slot after obtaining the page from disc or other storage media.

Paging and segmenting interrupt requests set bit-22 of the APR Control Register, the Memory Protection Flag. Therefore, an interrupt will take place on the channel specified by bits 33-35 of the APR Control

Register. A CONO order with bit 22 = 1 must be given by the interrupt program to the APR to reset this bit.

The flags and contents of PC (instruction producing the interrupt) are contained in Interrupt Summary Register 1. These - combined with the contents of Interrupt Summary Register 2 - should allow unique determination of the cause of the interrupt.

Hardware modifications have been made so that all instructions will sequence, as normal, to a time at which an interrupt strobe request may take place after a "fault" condition is detected. To insure that no modifications to core data are made, all memory references from the time of detecting the fault until the interrupt takes place (including accumulators/index registers) are forced to 57_8, the DATA SINK address.

An exception is when a failure occurs on a BLT order. The incrementing of the C(AC) is inhibited so that at interrupt time, the C(AC) reflect the location which gave difficulty.

The assumption that the referenced page is in core is basic to all I/O performed in User Mode. If failure does occur on the memory store portion of a DATAI, BLKI or CONI, the I/O cycles will take place with the resultant data stored in the DATA SINK address. If it is desired to maintain continuity, special software buffering techniques must be used to retain the data until the page is brought in (or assigned to) core. On DATAO or BLKO, the I/O cycles will be inhibited. For either BLKI or BLKO, the pointer must be decremented if the pointer word was successfully incremented and failure occurred on the data memory reference to maintain continuity.

The RELA instruction may be given in either Executive or User Mode. In User Mode either the address being relocated or the destination address may fail. If the address to be relocated produces a check condition, 57_8 will be stored in 57_8 (DATA SINK) and "Fetch Time" will be set in Interrupt Summary Register 2. Failure on the destination address will result in the relocated C(AC) being placed in location 57_8 and will set "Store Time".

In Executive Mode, failure to relocate the C(AC) will result in 57_8 being placed in the absolute effective address location. The "Fetch Time" flip-flop will be set in Interrupt Summary Register 2, along with other flip-flops giving the cause of failure, but CPA ILLEGAL ADDRESS will <u>not</u> be set so that an immediate interrupt attempt will <u>not</u> be made upon return to User Mode.

5.7 Segmenting and Paging Checks

Many checks are made at a hardware level to detect software and possible hardware errors. (Indicators are available to interrupt programs via the Interrupt Summary Registers, specified in Sections 4.3.3 and 4.3.4.)
These checks are defined here:

5.7.1 Parity Failure on Segment Number

Parity is checked (odd parity correct) on the segment number contained in the Address Base Register each time an address is relocated. It is also checked when access is made to the Descriptor Table to obtain page size information for FIRST ABR ACCESS, to obtain data to place in the Associative Memory upon a match failure, or to mark the FIRST WRITE condition into the Page Table. These three conditions will be included in the category of Descriptor Table access.

5.7.2 Descriptor Table Bounds Violation

The segment number is compared with the bounds contained in the Descriptor Base Register on Descriptor Table access. The bounds violation is produced when the designated segment number is strictly greater than the bounds.

5.7.3 Page Table Bounds Violation

The designated page number is compared with the segment bounds (contained in the Descriptor Table entry) on Page Table access. A bounds violation is produced when the page number is strictly greater than the specified bounds.

5.7.4 Segment Undefined

A segment number equal to 177_8 will produce this failure indication. This is checked on all relocation operations and when an Address Base Register is loaded with an LDR order. Since Address Base Registers and the Associative Memory are preset to segment number 177_8, access to a Base Register which has not been loaded will be detected by this mechanism.

5.7.5 Segment Overflow

An augmented address which exceeds the address that may be specified within a segment (dependent on page size) produces a

Segment Overflow indication (see Section 5.5). This is done by checking for 1's in the augmented address in bit positions:

a. 21 for 2^9 word pages,

b. 21, 22 and 23 for 2^7 pages, and by always detecting a carry in the adder or program counter from bit 21 to 20.

5.7.6 Segment Not in Core

On Descriptor Table access, bit 25 is set to Ø if a Page Table has not been defined to exist for the segment. Relocation is terminated, producing an interrupt.

5.7.7 Page Not in Core

Bit 12 (or 3Ø) is Ø when the Page Table is referenced; if the referenced page is not in core, producing an interrupt. Note that system programs must not remove a page from core whose relocation data is currently contained in the Associative Memory without terminating the user. No means exists for resetting this bit to Ø in the Associative Memory after it has been loaded as a 1.

5.7.8 Procedure Bit Not Enabled

Bit 13 (or 31) is set to 1 if the referenced page may be read as procedure for purposes of execution. Referencing a page at INSTRUCTION time with this bit reset to Ø produces an interrupt, setting "Procedure Bit Not Enabled".

5.7.9 Data Read bit Not Enabled

This bit is set if an attempt is made to access - at ADDRESS or Fetch Time - a page which is not enabled for data read.

5.7.10 Data Write Bit Not Enabled

This bit is set if an attempt is made to access at STORE time a page which is not enabled for data write.

5.7.11 Illegal Instruction in User Mode

LDR access to the Procedure Base Register or Descriptor Base Register or UNLR access to the Interrupt Summary Registers are denied in User Mode. Attempts to access these registers in User Mode will cause an interrupt setting this flag.

Second Edition

APPROVED	LAWRENCE RADIATION LABORATORY	ELECTRONICS ENGINEERING	UNIVERSITY OF CALIFORNIA	[illegible] Systems Report No. 2 Pt. XI
CHECKED		LIVERMORE, CALIFORNIA		
ORIG. [illegible] Ahrson	PDP-6 SEGMENTING AND PAGING SYSTEM			DATE 7/7/66
APP.	Revised System Description With Programming Notes			PAGE [illegible] OF [illegible] 10/66

5.7.12 Time of Failure

The time of the failure condition detected by the checks just listed, is recorded in bits 11-15 of Interrupt Summary Register 2.