SYSTEM OVERVIEW

OPERATING SYSTEMS BOOKS

Madnick, S.E. and Donovan, J.J.
Operating Systems, McGraw Hill, 1974

How operating systems are put together. For a person weak in thi

Brinch Hansen, P. Operating System Principles,
Prentice-Hall, 1973

How you should write op. systems, not how they are.

Organick, E.I., The Multics System: An Examination of
its Structure MIT Press, 1972

Primos is based on Multics; a good book, can be read on multiple levels; well structured.

MULTICS TECHNICAL REPORTS

MAC-TR-123 Introduction to Multics

also MAC-TR- Schedulers

FROM:

Laboratory for Computer Sciences
MIT
545 Technology Sq.
Cambridge, MA 02139
(617) 253-5894

Can get on their mailing list; once a year put out list of available works.

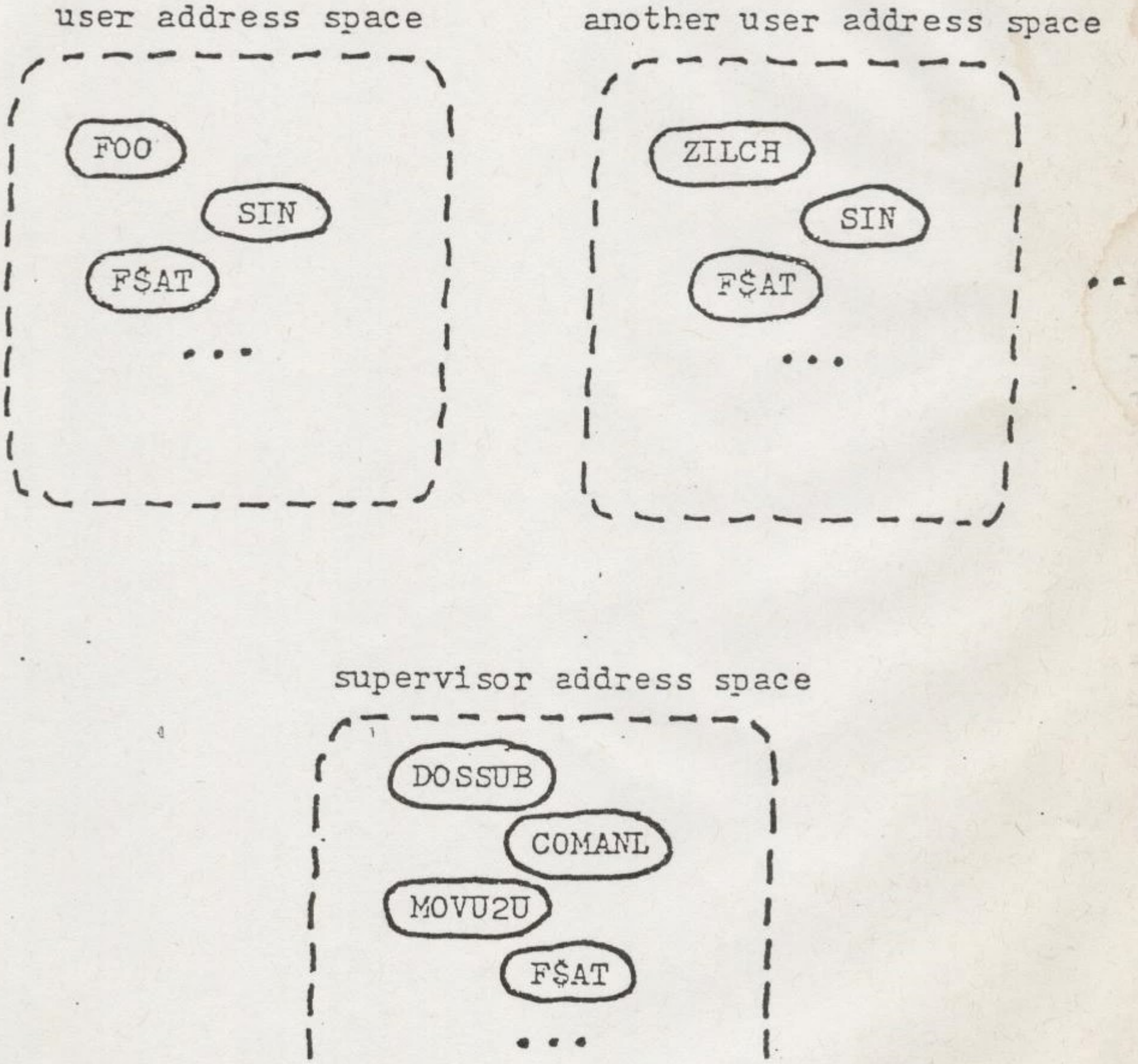
PRIME 350-750

SYSTEM ARCHITECTURE

The Prime 350-750 system embodies a number of novel architectural concepts which form the foundation for an efficient, powerful operating system: recursive/rentrant instruction set, firmware process dispatching, paged/segmented virtual memory, firmware stack management, and protection rings. Understanding these concepts and the way the software utilizes them is prerequisite to understanding Prime's product line today.

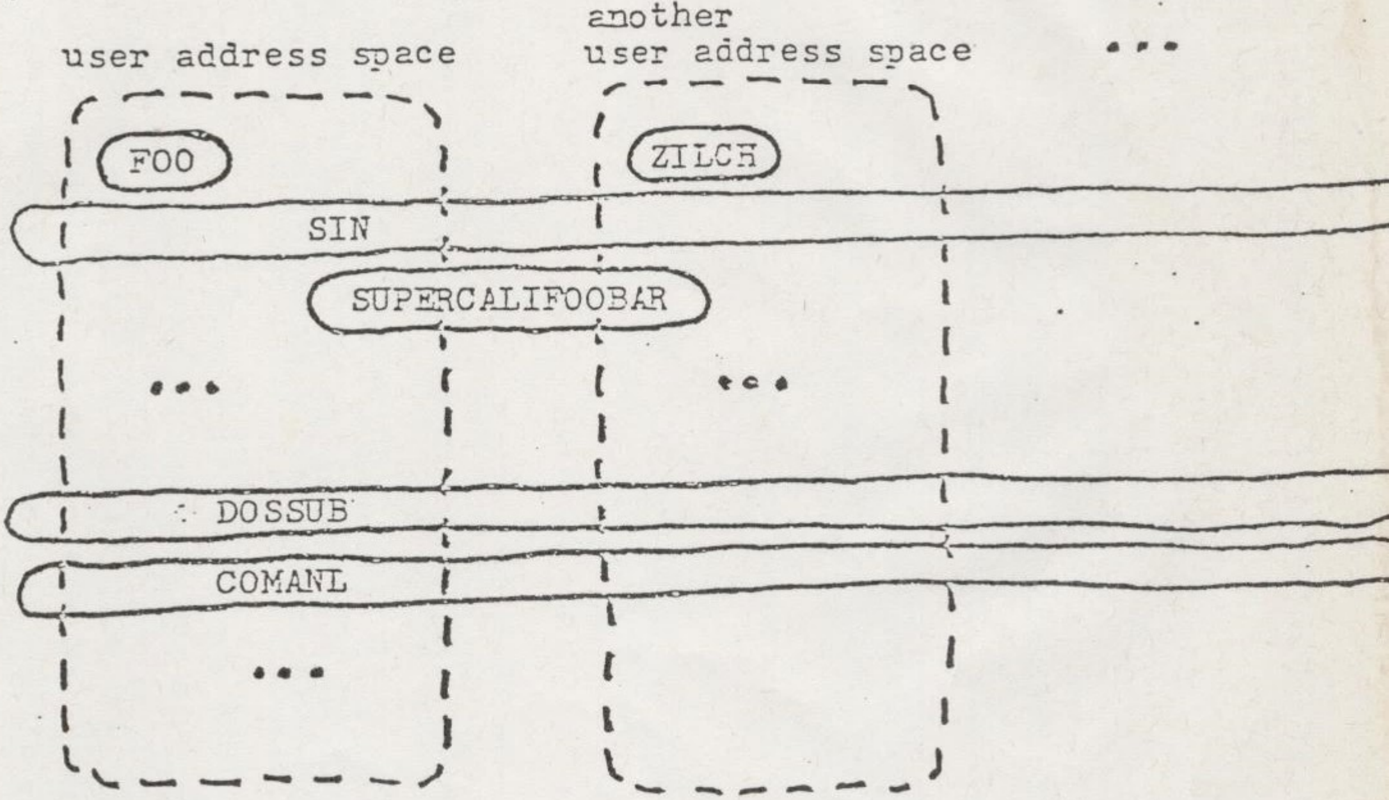
NON-EMBEDDED OPERATING SYSTEM

(PRIMOS III, OS/360)



EMBEDDED OPERATING SYSTEM

(PRIMOS IV, MULTICS)



~~F$AT~~

~~MOVU2U~~

ADVANTAGES OF
AN EMBEDDED OPERATING SYSTEM

- Efficient argument passing to the supervisor.
- Reentrant supervisor versus serially-reusable supervisor.
- User replaceability of supervisor components.

WHY NOT EMBED

- Protection hardware is inadequate.
- Instruction set is not reentrant.
- Address space is inadequate for sharing.

PAGING versus SEGMENTATION

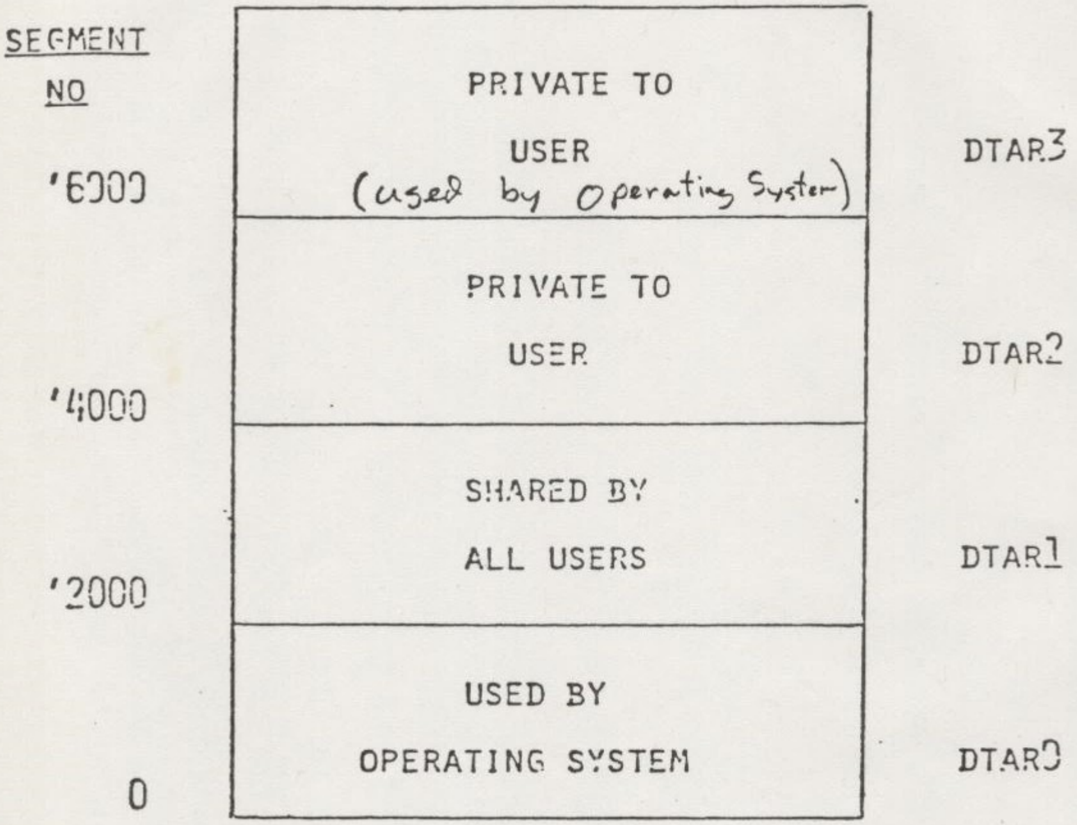
- PAGING is wholesaling of the physical address space.
  - Pages are uniform in size.
  - Paging solves the main-memory placement problem for the operating system.
  - Paging benefits the operating system, and is usually invisible to the user.

- SEGMENTATION is wholesaling of the virtual address space.
  - Segments are variable in size.
  - Segments hold modules (programs or data).
  - Segments facilitate address-space management (variable-sized modules; sharing).
  - Segments facilitate access control (sharing; protected subsystems).
  - Implied segment numbers shorten address fields and allow encapsulation of old programs. Can use R-mode in Seg. 4000
  - Segmentation benefits and is visible to the user.

- PAGING and SEGMENTATION can be combined in a system, to gain the benefits of both.

SEGMENTS ARE DIVIDED INTO 4 GROUPS OF 1024 ('2000)

- DESCRIPTOR TABLE ADDRESS REG
  (DTAR 0-3)

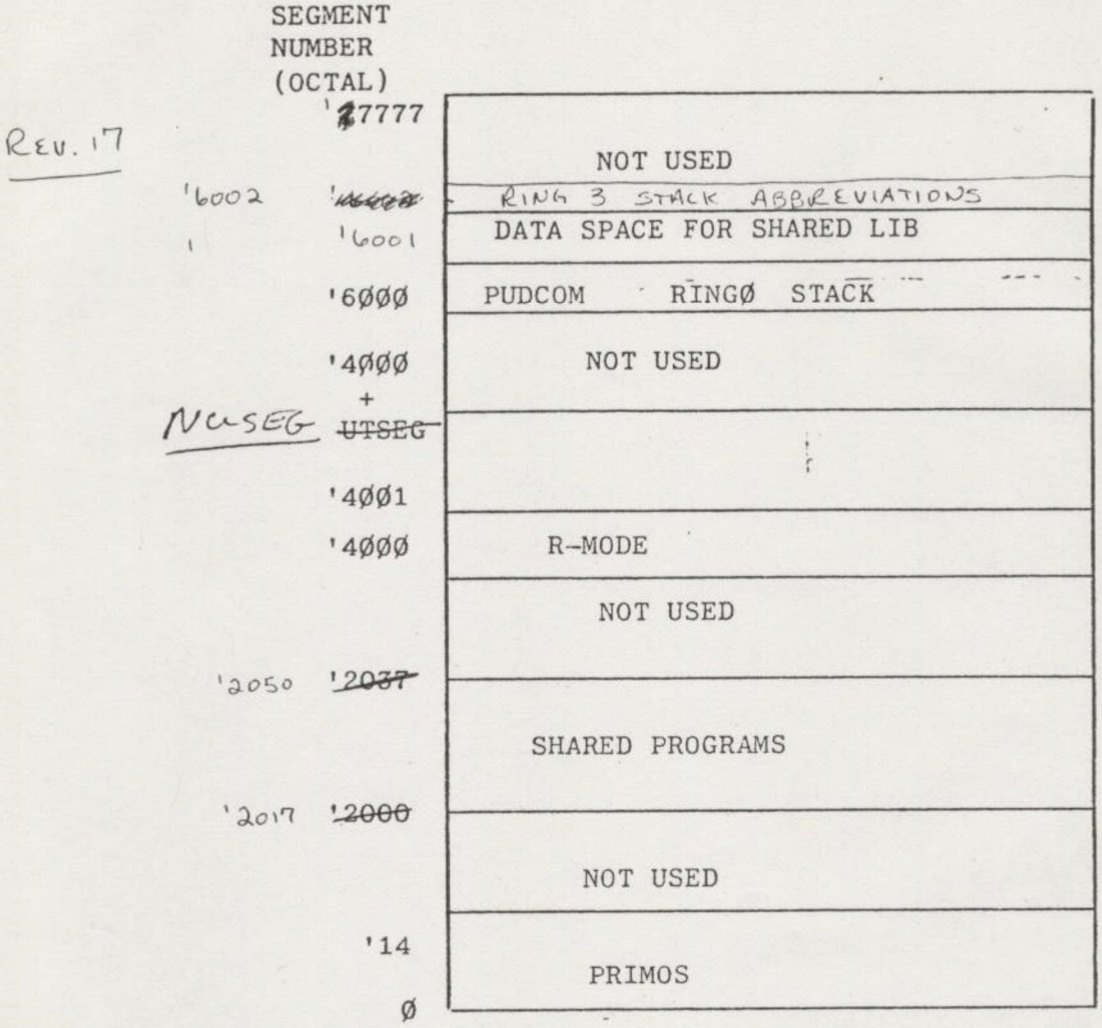


DTAR0 - USED BY OPERATING SYSTEM

DTAR1 - SHARED BY ALL USERS

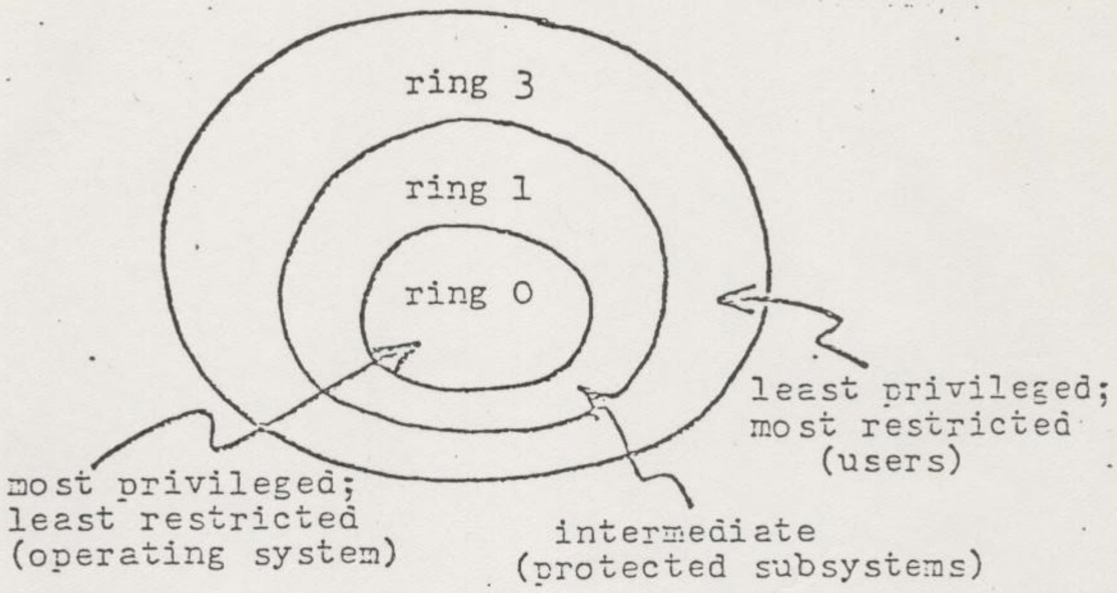
DTAR2 }
DTAR3 } - PRIVATE TO USER

A USER'S VIRTUAL MEMORY



PROTECTION RINGS

- Hierarchical domains of successively more restricted privilege.



- Modules live in rings, and processes visit them.

- Your privilege is determined by who you are (what segment table you use) and by what ring you are in (what module you are executing).

- Segment descriptor (32 bits):

1	3	3	3	22
F	A	B	C	P

F segment fault if set

P physical address of page table (22 bits)

A access allowed from ring 1: execute/read/write

B (reserved for access allowed from ring 2)

C access allowed from ring 3; execute/read/write

(all access is allowed from ring 0)

WEAKENING

- The ring from which access is made is carried along with every effective address computation.

  Space is provided for the ring-of-access in all base registers, in the field address registers, and in indirect words.

- The ring-of-access begins with the ring in which the process is executing (the ring field of the RP).

- The ring-of-access is then <u>weakened</u> by the ring field in any base register, field-address register, or indirect word used in the effective address calculation.

- The final weakened ring number is then used to select the allowed access privileges from the segment descriptor.

Draft for approval
Published: 1/23/66
Major Revision

All of this is not applicable to PRIMOS
(but may be at 19, 20... ?)

Identification

Protection of the Supervisor
R. Montrose Graham

Purpose

It is essential that certain supervisor procedures and data bases be totally inaccessable to a user. However, the supervisor must be callable by a user; and, when called, it must be able to access those protected segments which it needs to perform its function. Hence, a method of controlled entry to the supervisor is required, one which removes access restrictions for a group of segments as control passes to the supervisor. Further, it is desirable that the supervisor be protected from itself. Some segments of the supervisor are more sensitive than the others. Access to these segments by the rest of the supervisor should be controlled in the same manner as user access to the supervisor. This minimizes the chance of disaster in the event of minor machine errors and bugs in the supervisor itself. In addition, it aids in testing new supervisor modules. Finally, the same protection mechanism should be extendable for use by the users in such situations as an instructor's grading program and a student's solution, where the relation between programs is analogous to the supervisor-user relationship. The following paragraphs describe a framework in which all of these goals can be achieved.

Domains of Access, Rings, Walls

The segments of a process are divided into a number of mutually exclusive subsets, called *rings*. A segment <a>, is in one and only one ring. If we write <a> ϵ R(3) we mean that <a> is in ring 3. It is helpful to view these rings as annuli with the innermost ring being the hard core supervisor (see figure 1). The lines between rings are *walls*. The *domain of access* of segment <a> , D(a), is the union of the ring which contains <a> and all outer rings. In figure 1, D(a)=R(3) U R(4) (i.e., the union of ring 3 and ring 4). D(a) is the set of all segments which <a> may access. The complement of D(a), R(2) U R(1) in figure 1, is the set of segments to which <a> is denied any access. The hard core supervisor has access to all segments of the process. As control passes outward, access is denied for more and more segments, i.e., the domain of access gets smaller. When control is in R(i) we will say that the segments which are accessable are *unlocked* and those which are inaccessable are *locked*. Whenever control crosses a wall, the domain of access changes. Hence, when control passes from R(i) to R(i+1) all the segments in R(i) have to be locked and when control passes from R(i+1) to R(i) all the segments in R(i+1) have to be unlocked. Since all segments within a ring have the same domain of access, procedures in the same ring may freely call each other. In figure 1, <a> may call <b> and <y> . On the other hand, we want controlled entry to R(i) from R(i+1). There are a number of entry points to procedures in R(i), called gates, to which a procedure in an outer ring may legally transfer control. When control crosses the wall between R(i) and R(i+1) the segments of R(i) must be locked or unlocked depending upon the direction of crossing. In figure 1, suppose <a>|[ea] is a gate of R(3). If <d> calls <a>|[ea] the segments <a> ,. <b> ,..., <x> , and <y> have to be unlocked. If <a> then calls <h> the segments <a>, <b> ,..., <x> and <y>have to be locked since they are not in the domain of access of <h> . Thus, if the locking and unlocking is to be achieved automatically, crossing a wall in either direction must be detected. The procedure segments in each ring are, in general, normal slave procedures which use a stack. The contents of this stack needs to be protected in outer rings. Hence, each ring has its own stack segment which is a member of the ring. When a wall is crossed stacks must be switched, i.e., as control passes through a wall into ring i, the stack pointer is changed to point to the stack associated with ring i. In summary, when a wall is crossed, 1) the crossing has to be validated, 2) a number of segments have to be locked or unlocked, and 3) the stack has to be switched.

Crossing a Wall

Crossing a wall in either direction is detected by a fault. There is a distinct descriptor segment, D(i), associated with each ring, R(i). The contents of all the descriptor segments are identical, except possibly the access control bits, i.e., the kth descriptor in each D(i) refers to the same segment. When control is in R(i) the descriptor base register , DBR, points to D(i). The domain of access of a segment in R(i) is defined by the access control bits of the descriptors in D(i). Figure 2 shows the access control of the D(i) for the example in figure 1. When control is in R(i) only those procedures which are in R(i) are marked procedure in D(i). Any attempt to transfer control to a procedure not in R(i) results in a fault. In this fashion all crossings of a wall are detected. There are four different crossing situations:

1. Inward call; e.g., <d> calls <a>
2. Outward return; e.g., <a>returns to <d>
3. Outward call; e.g., <a> calls <h>
4. Inward return; e.g., <h>returns to <a>

Note: Dear U-code allows call outward, but not return inward!

Inward crossings are detected by a directed fault and outward crossings are detected by an attempt-to-execute-data fault. When a wall is crossed and control passes to R(i) the stack is switched and the DBR is set to point to D(i). This changing of effective descriptor segment accomplishes the locking or unlocking of the appropriate segments. Each of the four crossing situations is described in detail below.

Inward Call

If a directed fault occurs and the instruction which caused the fault is a transfer type (tra, tze, ... but not rtd) then an inward call is being attempted. An inward call is legal only if the location to which control is being transferred is a gate. The processor status when the fault occurs gives the number of the calling segment (e.g., d#) and the segment number and address of the entry point, (e.g., a#| ea). From this information it is determined to what rings d# and a# belong (in figure 1, d# ∈ R(4) and a# ∈ R(3)). Associated with each ring, R(i), is a gate list, G(i) (which can be hash coded). The gate list for R(i) contains a list of all gates to R(i) and the ring from which each gate may be entered. In the example, if the pair (a# | ea,4) is on G(3) then <d> may call <a>| [ea]. When it has been determined that this is a valid inward call to R(i), the stack is switched and the DBR is set to point to D(i). Execution of the faulting instruction is then completed.

Outward Return

If an attempt-to-execute-data fault occurs and the instruction causing the fault is an rtd, then an outward return is being attempted. The number of the segment to which return is being attempted (e.g., d#) is obtained from the machine conditions at the time of the fault. The ring number, R(i), of this segment is then determined. If the segment descriptor in G(i) is marked procedure, then the return is valid. In the example <d> is in R(4) and its descriptor in D(4) is marked procedure. Recall that a procedure is marked procedure in the descriptor segment of the ring to which it belongs, marked data in the descriptor segment of all inner rings, and marked directed fault in the descriptor segment of all outer rings. After it has been determined that this is a valid outward return a flag is set in the stack which indicates that control is passing outward from this ring via an outward return. Then the stack is switched and the DBR is set to point to D(i). Execution of the faulting instruction is then completed.

Outward Call

An outward call is being attempted when an attempt-to-execute-data fault occurs and the instruction causing the fault is a transfer type (tra, tze, ..., but not rtd). The outward call is validated in the same manner as the outward return. However, before the call can be completed, if the calling sequence includes arguments, the arguments must be moved into an area that is accessible by the procedure in the outer ring. Without making the rule that all arguments to an outward call must lie in an outer ring, which is undesirable, the caller may have indicated as an argument some location in a segment in the ring of the caller.

For example, if <a> calls <h> with two arguments one being in <y> and the second being in <z> then the argument in <y> must be moved to some segment which <h> may access. Therefore, before the call is completed all arguments which are not accessible by the called procedure will be moved into the stack belonging to the ring of the called procedure. Since there are a number of different types of arguments there are a number of different actions which may be required. The standard call provides for type information to be stored in the argument pointer (See Section BD.7.02). If the type code is 0, it is assumed that the argument pointer is pointing to one word of information. If the type code is non-zero it indicates the structure of the argument. The number of different types which will be handled properly on an outward call is restricted to those which are defined as part of the standard system module interfaces (See Section BB.2). Any of the data, specifiers, or dope for any of the arguments which lie in a segment which is not accessible to the called procedure will be moved into the stack corresponding to the ring of the called procedure. A new argument list will be constructed in which the argument pointers will point to the appropriate new location of all data. This argument list will also be placed in the stack of the called procedure. The location of the original argument list is saved in the stack of the caller for use when the called procedure returns (see below). In addition, the normal return point for this call is also saved for use in validating the return. A flag is set in the stack indicating that control is passing outward from this ring via an outward call. After this has been done the stack is switched, the DBR is properly set, and the faulting instruction is then completed.

Inward Return

If a directed fault occurs and the instruction which caused the fault is an rtd then an inward return is being attempted. The stack is switched first since it contains information which is needed to validate the inward return. The inward return is validated in the following fashion. The contents of the stack are examined to see if the last outward transfer of control from this ring was a call rather than a return. If it was a call the address to which control is now attempting to transfer is compared with the normal return point for the previous call. If they match the inward return is valid. If they do not match a check is made to see if any of the arguments of the call were label data. Any arguments which were label data represent possible alternate return points. These addresses are compared with the address to which control is now attempting to transfer. If a match is found then this is a valid inward return. If no match is found the return is invalid and appropriate error action is taken. When it is found that the inward return is valid, all arguments of the original outward call which had to be moved into the stack for accessibility are checked to see if they have been changed. Any arguments which have been changed by the called procedure must be moved back to their original position. If the original location of any of these arguments was in a read-only procedure a fault will occur during this process. This fault indicates the caller violated the read-only restriction of the argument and appropriate error action is taken at this point.

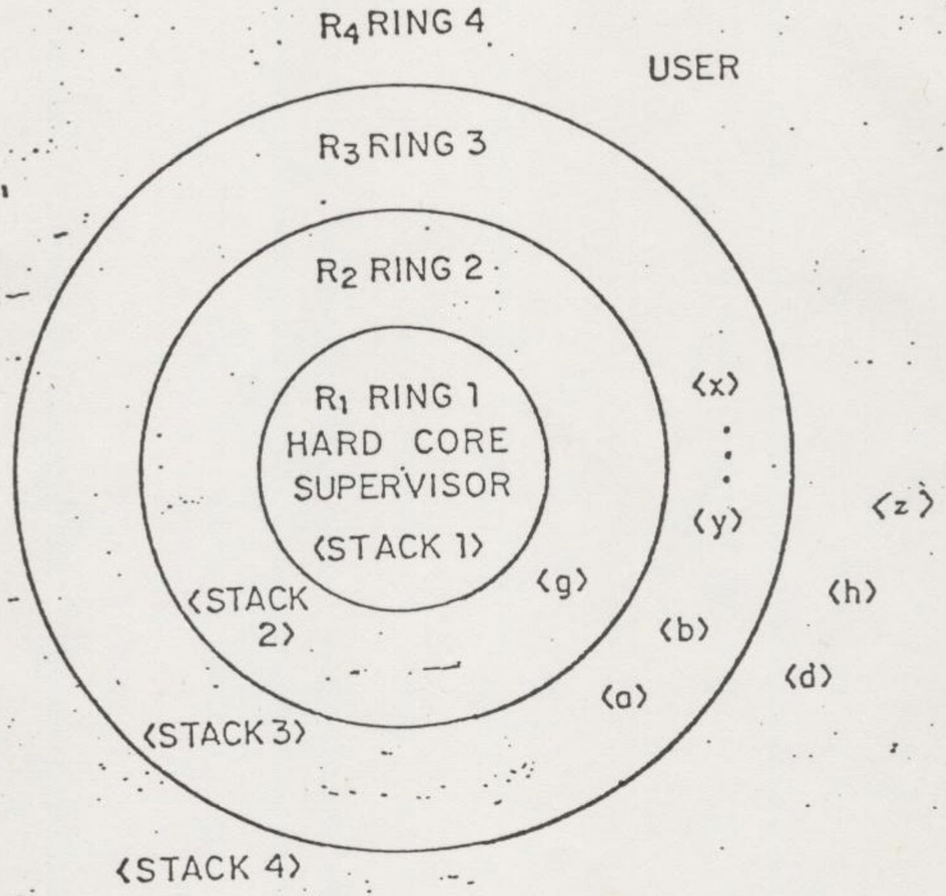


FIGURE 1: DIVISION OF THE SEGMENTS IN A PROCESS INTO SUBSETS, CALLED RINGS.

		D(4)	D(3)	D(2)
R(4)	<d>	proc slave access	data slave access	data slave access
	<h>	proc slave access	data slave access	data slave access
	<z>	data slave access	data slave access	data slave access
R(3)	<a>	directed fault	proc slave access	data slave access
	<b>	directed fault	proc slave access	data slave access
	<y>	master access only	data slave access	data slave access
	'''	''		
	<x>	master access only	data slave access	data slave access
R(2)	<g>	directed fault	directed fault	proc slave access

Figure 2: Access Controls in the D(i) for figure 1.

Operating Systems

B. Randell
Editor

The Multics Virtual Memory: Concepts and Design

A. Bensoussan, C.T. Clingen
Honeywell Information Systems, Inc.*
and
R.C. Daley
Massachusetts Institute of Technology†

As experience with use of on-line operating systems has grown, the need to share information among system users has become increasingly apparent. Many contemporary systems permit some degree of sharing. Usually, sharing is accomplished by allowing several users to share data via input and output of information stored in files kept in secondary storage. Through the use of segmentation, however, Multics provides direct hardware addressing by user and system programs of all information, independent of its physical storage location. Information is stored in segments each of which is potentially sharable and carries its own independent attributes of size and access privilege.

Here, the design and implementation considerations of segmentation and sharing in Multics are first discussed under the assumption that all information resides in a large, segmented main memory. Since the size of main memory on contemporary systems is rather limited, it is then shown how the Multics software achieves the effect of a large segmented main memory through the use of the Honeywell 645 segmentation and paging hardware.

Key Words and Phrases: operating system, Multics, virtual memory, segmentation, information sharing, paging, memory management, memory hierarchy

CR Categories: 4.30, 4.31, 4.32

1. Introduction

In the past few years several well-known systems have implemented large virtual memories which permit the execution of programs exceeding the size of available core memory. These implementations have been achieved by demand paging in the Atlas computer [11], allowing a program to be divided physically into pages only some of which need reside in core storage at any one time, by segmentation in the B5000 computer [15], allowing a program to be divided logically into segments, only some of which need be in core, and by a combination of both segmentation and paging in the Honeywell 645 [3, 12] and the IBM 360 67 [2] for which only a few pages of a few segments need be available in core while a program is running.

As experience has been gained with remote-access, multiprogrammed systems, however, it has become apparent that, in addition to being able to take advantage of the direct addressibility of large amounts of information made possible by large virtual memories, many applications also require the rapid but controlled sharing of information stored on-line at the central facility. In Multics (*Mult*iplexed *I*nformation and *C*omputing *S*ervice) segmentation provides a generalized basis for the direct accessing and sharing of on-line information by satisfying two design goals: (1) it must be possible for all on-line information stored in

Minor revision of a paper presented at an ACM Symposium on Operating System Principles, Princeton University, October 20–22, 1969. Work reported herein was supported (in part) by Project MAC, an MIT research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(1). *Honeywell Information Systems, Inc., Cambridge, MA 02142. †Cambridge, MA 02142.

the system to be addressed directly by a processor and hence referenced directly by any computation; (2) it must be possible to control access, at each reference, to all on-line information in the system.

The fundamental advantage of direct addressibility that information copying is no longer mandatory. ce all instructions and data items in the system are processor-addressible, duplication of procedures and data is unnecessary. This means, for example, that core ages of programs need not be prepared by loading and binding together copies of procedures before execution; instead, the original procedures may be used ectly in a computation. Also, partial copies of data files need not be read, via requests to an I/O system, into core buffers for subsequent use and then returned, means of another I/O request, to their original cations; instead the central processor executing a computation can directly address just those required ata items in the original version of the file. This kind access to information promises a very attractive reduction in program complexity for the programmer.

If all on-line information in the system may be ddressed directly by any computation, it becomes imperative to be able to limit or control access to this information both for the self-protection of a computation from its own mishaps, and for the mutual protection of computations using the same system hardware facilities. Thus it becomes desirable to compartmentalize package all information in a directly-addressible emory and to attach access attributes to these information packages describing the fashion in which ach user may reference the contained data and procedures. Since all such information is processor-addressible, the access attributes of the referencing user must be enforced upon each processor reference any information package.

Given the ability to directly address all on-line information in the system, thereby eliminating the eed for copying data and procedures, and given the bility to control access to this information, controlled sharing among several computations then follows as a atural consequence.

In Multics, segments are packages of information which are directly addressed and which are accessed in a controlled fashion. Associated with each segment is set of access attributes for each user who may access the segment. These attributes are checked by hardware upon each segment reference by any user. Furthermore all on-line information in a Multics installation can be directly referenced as segments while in other systems most on-line information is referenced as files.

This paper discusses the properties of an "idealized" Multics memory comprised entirely of segments erenced by symbolic name, and describes the simulation of this idealized memory through the use of both pecialized hardware and system software. The result of this simulation is referred to as the Multics virtual memory. Although the Multics virtual memory has been discussed elsewhere [3, 6, 7] at the conceptual level or in its earlier forms, the implementation presented here represents a mechanism resulting from several consecutive implementations leading to an effective realization of the design goals.

2. Segmentation

A basic motivation behind segmentation is the desire to permit information sharing in a more automatic and general manner than provided by nonsegmented systems. Sharing must be accomplished without duplication of information and access to the shared information must be controlled not only in secondary memory but also in main memory.

In most existing systems that provide for information sharing, the two requirements mentioned above are not met. For example, in the CTSS system [5], information to be shared is contained in files. In order for several users to access the information recorded in a file, a *copy* of the desired information is placed in a buffer in each user's core image. This requires an explicit, programmer-controlled I/O request to the file system, at which time the file system checks whether the user has appropriate access to the file. During execution, the user program manipulates this copy and not the file. Any modification or updating is done on the copy and can be reflected in the original file only by an explicit I/O request to the file system, at which time the file system determines whether the user has the right to change the file.

In nonsegmented systems, the use of core images makes it nearly impossible to control access to shared information in core. Each program in execution is assigned a logically contiguous, bounded portion of core memory or paged virtual memory. Even if the nontrivial problem of addressing the shared information in core were solved, access to this information could not be controlled without additional hardware assistance. Each core image consists of a succession of anonymous words that cannot be decomposed into the original elementary parts from which the core image was synthetized. These different parts are indistinguishable in the core image; they have lost their identity and thereby have lost all their attributes, such as length, access rights, and name. As a consequence, nonsegmented hardware is inadequate for controlled sharing in core memory. Although attempts to share information in core memory have been made with nonsegmented hardware, they have resulted in each instance being a special case which must be preplanned at the supervisory level. For example, if all users are to share a compiler in main memory, it is imperative that none of them be able to alter the part of main memory where the compiler resides. The hardware "privileged" mode used by the supervisor is often the only means of protecting shared information in main memory. In order to protect the shared compiler, it is made accessible only in this privileged mode. The compiler can no longer be regarded as a user procedure; it has to be accessed through a supervisor call like any other part of the supervisor, and must be coded to respect any conventions which may have been established for the supervisor.

In segmented systems, hardware segmentation can be used to divide a core image into several parts, or segments [10]. Each segment is accessed by the hardware through a segment descriptor containing the segment's attributes. Among these attributes are access rights that the hardware interprets on each program reference to the segment for a specific user. The absolute core location of the beginning of a segment and its length are also attributes interpreted by the hardware at each reference, allowing the segment to be relocated anywhere in core and to grow and shrink independently of other segments. As a result of hardware checking of access rights, protection of a shared compiler, for example, becomes trivial since the compiler can reside in a segment with only the "execute" attribute, thus permitting users to execute the compiler but not to change it.

In most segmented systems, a user program must first call the supervisor to associate a segment descriptor with a specific file before the program can directly access the information in the file. If the number of files the user program must reference exceeds the number of segment descriptors available to the user, the user program is forced to call the supervisor again to free segment descriptors currently in use so that they can be reused to access other information. Furthermore, if the number of segment descriptors is insufficient to provide simultaneous direct access to each distinct file required by this program, the user must then provide for some means of buffering this information. Buffering, of course, requires that information from more than one file be copied and coalesced with other distinctly different information having potentially different attributes. Once the information is copied and merged, the identity of the original information is lost, thus making it impossible for the information to be shared with other user programs. In addition, this form of user-controlled segment descriptor allocation and buffering of information requires a significant amount of preplanning by the user.

In Multics, the number of segment descriptors available to each computation is sufficiently large to provide a segment descriptor for each file that the user program needs to reference in most applications. The availability of a large number of segment descriptors to each computation makes it practical for the Multics supervisor to associate segment descriptors with files upon first reference to the information by a user program, relieving the user from the responsibility of allocating and deallocating segment descriptors. In addition, the relatively large number of segment descriptors eliminates the need for buffering, allowing the user program to operate directly on the original information rather than on a copy of the information. In this way, all information retains its identity and independent attributes of length and access privilege regardless of its physical location in main memory or on secondary storage. As a result, the Multics user no longer uses files; instead he references all information as segments, which are directly accessible to his programs.

To Multics users, all memory appears to be composed of a large number of independent linear core memories, each associated with a descriptor. A user program can create a segment by issuing a call to the supervisor, giving, as arguments, the appropriate attributes such as symbolic segment name, name of each user allowed to access the segment with his respective access rights, etc. The supervisor then finds an unused descriptor where it stores the segment attributes. The segment having been created, the user program can now address any word of the corresponding linear memory by the pair (name, *i*) where "name" is the symbolic name of the segment and "*i*" is the word number in the linear memory. Furthermore, any other user can reference word number *i* of this segment also by the pair (name, *i*) but he can access it only according to the access rights he was given by the creator and which are recorded in the descriptor. Combinations of the "read," "write," "execute" and "append" access rights [6] are available in Multics.

A simple representation of this memory, referred to as the Multics idealized memory, is shown in Figure 1.

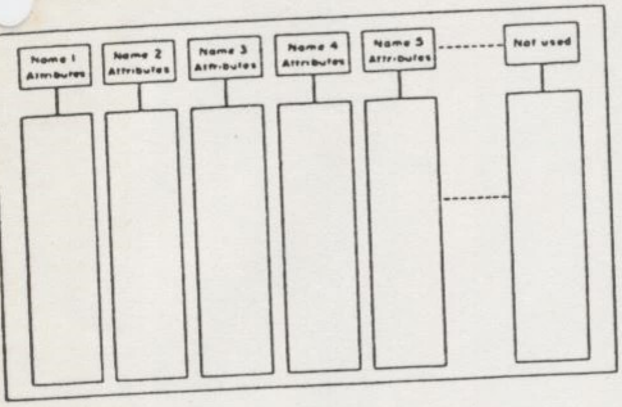
3. Paging

In a system in which the maximum size of any segment was very small compared to the size of the entire core memory, the "swapping" of complete segments into and out of core would be feasible. Even in such a system, if all segments did not have the same maximum size, or had the same maximum size but were allowed to grow from initially smaller sizes, there remains the difficult core management problem of providing space for segments of different sizes. Multics, however, provides for segments of sufficient maximum size so that only a few can be entirely core-resident at any one time. Also, these segments can grow from any initial size smaller than the maximum permissible size.

By breaking segments into equal-size parts called *pages* and providing for the transportation of individual pages to and from core as demand dictates, the disadvantages of fragmentation are incurred, as explained by Denning [9]. However, several practical problems encountered in the implementation of a segmented virtual memory are solved.

First, since pages are all of equal size, space allocation is immensely simplified. The problems of "com-

1. Multics idealized memory.



pacting" information in core and on secondary storage, characteristic of systems dealing with variable-sized segments or pages, are thereby eliminated.

Second, since only the referenced page of a segment need be in core at any one instant, segments need not be small compared to core memory.

Third, "demand paging" permits advantage to be taken of any locality of reference peculiar to a program by transporting to core only those pages of segments which are currently needed. Any additional overhead associated with demand paging should of course be weighed against the alternative inefficiencies associated with dedicating core to entire segments which must be swapped into core but which may be only partly referenced.

Finally, demand paging allows the user a greater degree of machine independence in that a large program designed to run well in a large core memory configuration will continue to run at reduced performance on smaller configurations.

4. The Multics Virtual Memory

Multics simulates the idealized memory, represented in Figure 1, using the segmentation and paging features of the 645 assisted by the appropriate software features. The result of the simulation is referred to as the "Multics Virtual Memory." The user can keep a large number of segments in this memory and reference them by symbolic name; upon first reference to a segment, the supervisor automatically transforms the symbolic name into the appropriate hardware address which is directly used by the processor for subsequent references.

The remainder of this paper explains the addressing mechanism in the 645 and describes how the Multics supervisor simulates the Multics idealized memory.

5. The Honeywell 645 Processor

The features of the 645 processor which are of interest for the implementation of the Multics virtual memory are segmentation and paging.

5.1 Segmentation

Any address in the 645 processor consists of a pair of integers [s, i]. "s" is called the *segment number*; "i" the index within the segment. The range of "s" and "i" is 0 to $2^{18} - 1$. Word [s, i] is accessed through a hardware register which is the sth word in a table called a *descriptor segment* (DS). The descriptor segment is in core memory and its absolute address is recorded in a processor register called a *descriptor base register* (DBR). Each word of the DS is called a *segment descriptor word* (SDW); the sth SDW will be referred to as SDW(s). See Figure 2.

The DBR contains the values:

DBR·core which is the absolute core address of the DS.

DBR·L which is the length of the DS.

Segment descriptor word number "s" contains the values:

SDW(s)·core which is the absolute core address of the segment s.

SDW(s)·L which is the length of the segment s.

SDW(s)·acc which describes the access rights for the segment.

SDW(s)·F which is the "missing segment" switch.

A simplified version of the algorithm used by the processor to access the word whose address is [s, i] follows (see Figure 2):

If DBR·L $<$ s, generate a trap, or "fault" to the supervisor.

Access SDW(s) at absolute location DBR·core + s.

If SDW(s)·F = ON, generate a *missing segment fault*.

If SDW(s)·L $<$ i, generate a fault.

If SDW(s)·acc is incompatible with the requested operation, generate a fault.

Access the word whose absolute address is SDW(s)·core + i.

5.2 Paging

The above description assumes that segments are not paged; in fact, paging is implemented in the 645 hardware. In the Multics implementation, all segments are paged and the page size is always 1,024 words.

Element "i" of a segment is the w^{th} word of the p^{th} page of the segment, "w" and "p" being defined by

$$\begin{cases} w = i \bmod 1{,}024 \\ p = (i - w)/1{,}024 \end{cases}$$

Each segment is referenced by a processor through a *page table* (PT). The PT of a segment is an array of

Fig. 2. Hardware segmentation in the Honeywell 645.

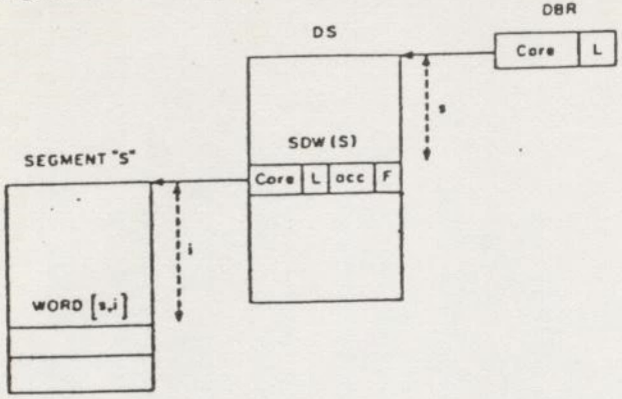
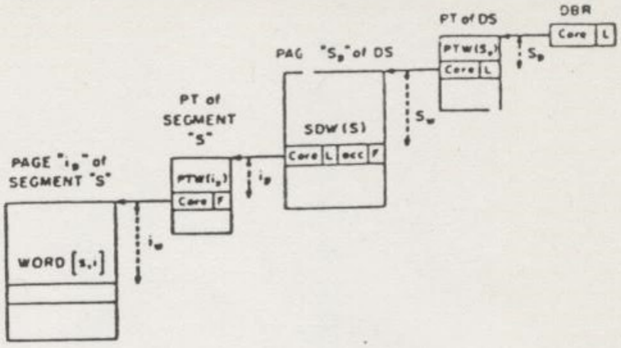


Fig. 3. Hardware segmentatic · and paging in the Honeywell 645.



physically contiguous words in core memory. Each element of this array is called a *page table word* (PTW). Page table word number p contains:

PTW(p)·core which is the absolute core address of page number p.

PTW(p)·F which is the "missing page" switch.

The meaning of DBR·core and SDW(s)·core is now:

DBR·core = Absolute core address of the PT of the descriptor segment.

SDW(s)·core = Absolute core address of the PT of segment number s.

A simplified version of the algorithm used by the processor to access the word whose address is [s, i] is as follows (see Figure 3):

If DBR·L $<$ s, generate a fault.
Split s into the page number s_p and word number s_w .
Access PTW(s_p) at absolute location
  DBR·core + s_p .
If PTW(s_p)·F = ON, generate a *missing page fault*.
Access SDW(s) at absolute location
  PTW(s_p)·core + s_w .
If SDW(s)·F = ON, generate a missing segment fault.
If SDW(s)·L $<$ i, generate a fault.
If SDW(s)·acc is incompatible with the requested operation, generate a fault.
Split i into the page number i_p and word number i_w .
Access PTW(i_p) at absolute location
  SDW(s)·core + i_p .
If PTW(i_p)·F = ON, generate a missing page fault.
Access the word whose absolute location is
  PTW(i_p)·core + i_w .

In order to reduce the number of processor references to core storage while performing this algorithm, each processor has a small, high-speed *associative memory* [12] automatically maintained so as to always contain the PTW's and SDW's most recently used by the processor. The associative memory significantly reduces the number of additional memory requests required during address preparations.

STLB on Prime

6. Multics Processes and the Multics Supervisor

A process is generally understood as being a program in execution. A process is characterized by its state-word defining, at any given instant, the history resulting from the execution of the program. It is also characterized by its *address space*. The address space of a process is the set of processor addresses that the process can use to reference information in memory. In Multics, any information that a process can reference by an address of the form (segment number, word number) is said to be in the address space of the process. There is a one-to-one correspondence between Multics processes and address spaces. Each process is provided with a private descriptor segment which maps segment numbers into core memory addresses and with a private table which maps symbolic segment names into segment numbers. This table is called the Known Segment Table (KST).

The Multics supervisor could have been written so as not to use segment addressing of course; but organizing the supervisor into procedures and data segments permits one to use, in the supervisor, the same conventions that are used in user programs. For instance, the call-save-return conventions [7] made for user programs can be used by the supervisor; the standard way to manufacture pure procedures in a user program is also used extensively in the supervisor. A less visible advantage of segmentation of the supervisor is that some supervisory facilities provided for the management of user segments can also be applied to supervisor segments; for example, the demand paging facility designed to automatically load pages of user segments can also be used to load pages of supervisor segments. As a result, a large portion of the supervisor need not be permanently in core.

Unlike most supervisors, the Multics supervisor does not operate in a dedicated process or address space. Instead, the supervisor procedure and data segments are shared among all Multics processes. Whenever a new process is created, its descriptor segment is initialized with descriptors for all supervisor segments allowing the process to perform all of the basic supervisory functions for itself. The execution of the supervisor in the address space of each process facilitates communication between user procedures and supervisor procedures. For example, the user can call a supervisor procedure as if he were calling a normal user procedure. Also, the sharing of the Multics supervisor facilitates simultaneous execution, by several processes, of supervisory functions, just as the sharing of user procedures facilitates the simultaneous execution of functions written by users.

Since supervisor segments are in the address space of each process, they must be protected against unauthorized references by user programs. Multics provides the user with a ring protection mechanism [13] which segregates the segments in his address space into several sets with different access privileges. The Multics supervisor takes advantage of the existence of this mechanism and uses it, rather than some other special mechanism to protect itself.

7. Segment Attributes

7.1 Directory Hierarchy

The name of a segment and its attributes are associated in a catalogue. Conceptually this catalogue consists of a table with one entry for each segment in the system. An *entry* contains the name of the segment and all its attributes: length, memory address, list of users allowed to use the segment with their respective access rights, date and time the segment was created, etc.

In Multics, this catalogue is implemented as several segments, called directories, organized into a tree structure. A *segment name* is a list of subnames reflecting the position of the entry in the tree structure, with respect to the beginning, or root directory (ROOT) of the tree. By convention, subnames are separated by the character ">". Each subname is called an *entryname* and the list of entrynames is called a *pathname*. An entryname is unique in a given directory and a pathname is unique in the entire directory hierarchy. Because of its property of uniquely identifying a segment in the directory hierarchy, the pathname has been chosen as the symbolic name by which the Multics user must reference a segment. There are two types of directory entries, branches and links. A *branch* is a directory entry which contains all attributes of a segment while a *link* is a directory entry which contains the pathname of another directory entry. A more detailed description of the directory hierarchy and of the use of links is given by Daley and Neumann [6].

7.2 Operations on Segment Attributes

Supervisor primitives perform all operations on segment attributes. There is a set of primitives available to the user which allow him, for example, to create a segment, delete a segment, change the entryname of a segment, directory entry, change the access rights of a segment, list the segment attributes contained in a directory, etc.

Creating a segment whose pathname is ROOT > A > B > C (see Figure 4) consists basically of the following steps:

Check that entryname C does not already exist in the directory ROOT > A > B.

Allocate space for a new branch in directory ROOT > A > B.

Store in the branch the following items:

The entry name C.

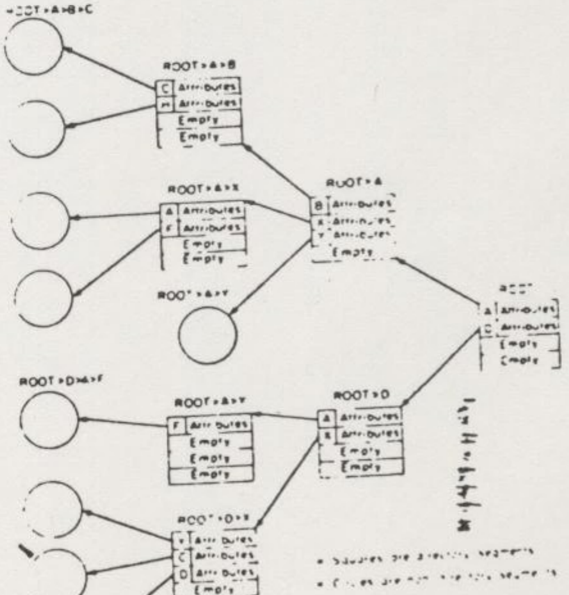
The segment length, initialized to zero.

The access list, given by the creator.

The segment map, consisting of an array of secondary memory addresses, one for each page of the segment. The maximum length of a segment in Multics being 64 pages, the segment map for any segment contains 64 entries. Since the segment length is still zero, each entry of the segment map is initialized with a "null" address, showing that no secondary memory has been assigned to any potential page of the segment.

The segment status "inactive," meaning that there is no page table for this segment. The segment status, which may be either "active" or "inactive" is indicated by the *active switch*.

Fig. 4. Directory hierarchy.



8. Segment Accessing

Although the creation of a segment initializes its attributes, additional supervisor support is required to make the segment accessible to the processor when a user program references the segment by symbolic name.

8.1 Symbolic Addressing Conventions

The pathname is the only symbolic name by which a segment can be uniquely identified in the directory hierarchy. However, for user convenience, the system provides a facility whereby a user can reference a segment from his program using only the last entryname of the segment's pathname and supplying the rest of the pathname according to system conventions. This last entry name is called the *reference name.*

When a process executes an instruction which attempts to access a segment by means of its reference name, the Multics dynamic linking facility [7] is automatically invoked. The dynamic linker determines the missing part of the pathname according to the above-mentioned system conventions. These conventions are called *search rules* and may be regarded as a list of directories to be searched for an entryname matching the specified reference name. When this entryname is found in a directory, the directory pathname is prefixed to the reference name yielding the required pathname. The dynamic linker, using the "Make Known" module (Section 8.2), then obtains a segment number by which the referenced segment will be accessed. Finally it transforms the reference name into this segment number so that all subsequent executions of the instruction in this process access the segment directly by segment number. Further details are given by Daley and Dennis [7].

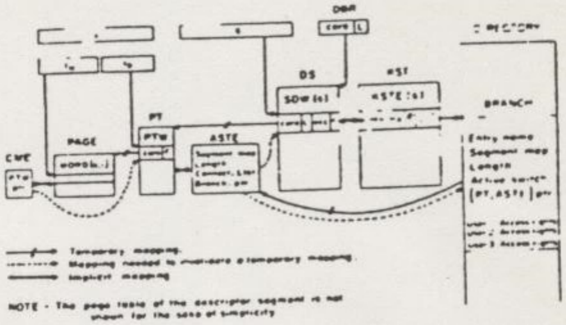
8.2 Making a Segment Known to a Process

Each time a segment is referenced in a process by its pathname, either explicitly or as the result of the evaluation of a reference name by the dynamic linking facility, the pathname must be translated into a segment number in order to permit the processor to address the segment for this process. This translation is done by the supervisor using the KST associated with the process. The KST is an array organized such that entry number "*s*", KSTE(*s*), contains the pathname associated with segment number "*s*". See Figure 5.

If the association (pathname, segment number) is found in the KST of the process, the segment is said to be *known* to the process and the segment number can be used to reference the segment.

If the association (pathname, segment number) is not found in the KST, this is the first reference to the segment in the process and the segment must be made known. A segment is made known by assigning an unused segment number "*s*" in the process and by recording the pathname in KSTE(*s*) to establish the pair (pathname, segment number) in the KST of the process. The directory hierarchy is also searched for this pathname and a pointer to the corresponding branch is entered in KSTE(*s*) for later use (Section 8.3.).

Fig. 5. Basic tables used to implement the Multics virtual memory.


DBR
DS
SDW(s)
AST
ASTE(s)
BRANCH
PT
PTW
PAGE
ASTE
CME
Entry name
Segment map
Length
Active switch
(PT,ASTE) ptr
Temporary mapping
Mapping needed to maintain a temporary mapping
Implicit mapping
NOTE - The page table of the descriptor segment is not shown for the sake of simplicity

The per-process association of pathname and segment number is used in the Multics system because it is impossible to assign a unique segment number to each segment. The reason is that the number of segments in the system will nearly always be larger than the number of segment numbers available in the processor.

When a segment is made known to a process by segment number "*s*," its attributes are not placed in SDW(*s*) of the descriptor segment of that process. SDW(*s*) having been initialized with the missing segment switch ON, the first reference in this process to that segment by segment number "*s*" will cause the processor to generate a trap. In Multics this trap is called a "missing segment fault" and transfers control to a supervisor module called the segment fault handler.

8.3 The Segment Fault Handler

When a missing segment fault occurs, control is passed to the segment fault handler to store the proper segment attributes in the appropriate SDW and set the missing segment switch OFF in the SDW.

These attributes, as shown in Figure 3, consist of the page table address, the length of the segment, and the access rights of the user with respect to the segment. The information initially available to the supervisor upon occurrence of a missing segment fault is the segment number "s."

The only place where the needed attributes can be found is in the branch of the segment. Using the segment number "*s*", the supervisor can locate the KST entry associated with the faulting segment; it can then find the required branch since a pointer to the branch has been stored in the KST entry when the segment was made known to this process (Section 8.2).

Using the active switch (Figure 5) in the branch, the supervisor determines whether there is a page table for this segment. Recall that this switch was initialized in the branch at segment creation time. If there is no page table, one must be constructed. A portion of core memory is permanently reserved for page tables. All page tables are of the same length and the number of page tables is determined at system initialization.

The supervisor divides page tables into two lists: the used list and the free list. Manufacturing a page table (PT) for a segment could consist only of selecting a PT from the free list, putting its absolute address in the branch and moving it from the free to the used list. If this were actually done, however, the servicing of each missing page fault would require access to a branch since the segment map containing secondary storage addresses is kept there (Figure 5). Since it is impractical for all directories to permanently reside in core, page fault handling could thereby require a secondary storage access in addition to the read request required to transport the page itself into core. Although this mechanism works, efficiency considerations have led to the "activation" convention between the segment fault handler and the page fault handler.

Activation. A portion of core memory is permanently reserved for recording attributes needed by the page fault handler, i.e. the segment map and the segment length. This portion of core is referred to as the *active segment table* (AST). There is only one AST in the system and it is shared by all processes. The AST contains one entry (ASTE) for each PT. A PT is always associated with an ASTE, the address of one implying the address of the other. They may be regarded as a single entity and will be referred to as the (PT, ASTE) of a segment. The used list and free list mentioned above are referred to as the (PT, ASTE) *free list* and the (PT, ASTE) *used list*.

A segment which has a (PT, ASTE) is said to be *active*. Being active or not active is an attribute of the segment and is recorded in the branch using the active switch.

When the active switch is ON, both the segment map and the segment length are no longer in the branch but are to be found in the segment's (PT, ASTE) whose address was recorded in the branch during "activation" of the segment.

To activate a segment, the supervisor must:

Find a free (PT, ASTE). (Assume temporarily that at least one is available).

Move the segment map and the segment length from the branch into the ASTE.

Set the active switch ON in the branch.

Record the pointer to (PT, ASTE) in the branch.

By pairing an ASTE with a PT in core, the segment fault handler has guaranteed that all segment attributes needed by the page fault handler are core-resident, permitting more efficient page fault servicing.

Connection. Once the segment is active, the corresponding SDW must be "connected" to the segment. To connect the SDW to the segment the supervisor must:

Get the absolute address of the PT, using the (PT, ASTE) pointer kept in the branch, and store it in SDW.

Get the segment length from the ASTE and store it in the SDW.

Get the access rights for the user from the branch and store them in the SDW.

Turn off the missing segment switch in the SDW.

Having defined activation and connection, segment fault handling can now be summarized as:

Use the segment number *s* to access the KST entry.

Use the KST entry to locate the branch.

If the active switch in the branch is OFF, activate the segment.

Connect the SDW.

Note that the active switch and the (PT, ASTE) pointer in the segment branch "automatically" guarantee segment sharing in core since all SDW's describing a given segment will point to the same PT.

Once the segment and its SDW have been connected, the hardware can access the appropriate page table word. If the page is not in core, a missing page fault occurs, transferring control to the supervisor module called the page fault handler.

8.4 The Page Fault Handler

When a page fault occurs the page fault handler is given control with the PT address and the page number of the faulting page. The information needed to bring the page into core memory is the address of a free block of core memory into which the page can be moved and the address of the page in secondary memory. The term *page frame* is also used to denote a block of core memory which holds a page of information [9].

A free block of core must be found. This is done by using a data base called the *core map*. The core map is an array of elements called *core map entries* (CME). The n^{th} entry contains information about the n^{th} block of core (the size of all blocks is 1,024 words). The supervisor divides this core map into two lists; the *core map used list* and the *core map free list*.

The job of the page fault handler consists of the following steps:

Find a free block of core and remove its core map entry from the free list. (Assume temporarily that the free list is not empty.)

Access the ASTE associated with the PT and find the address in secondary memory of the missing page.

If this address is a "null" address, initialize the block of core with zeros and update the segment length in the ASTE; this action is only taken the first time the page is referenced since the segment was created and provides for the automatic growing of segments. Otherwise issue an I O request to move the page from secondary memory into the free block of core and wait for completion of the request via a call to the "traffic controller" [14] which is responsible for processor multiplexing.

Store the core address in the PTW, remove the fault from the PTW, and place the core map entry in the used list.

8.5 Page Multiplexing

There are many more pages in virtual memory than there are blocks of core in the real memory; therefore, these blocks must be multiplexed among all pages. In the description of page fault handling it was assumed that a free block of core was always available. In order to insure that this is nearly always true, the page fault handler, upon removing a free block from the core map free list, examines the number of remaining free list entries; if this number is less than a preset minimum value, a page removal mechanism is invoked a sufficient number of times to ensure a nonempty core map free list in all but the most unusual cases. A nonempty core map free list eliminates waiting for page removal during the handling of a missing page fault.

To get a free block of core, the page removal mechanism may have to move a page from core to secondary memory. This requires: (a) an algorithm to select a page to be removed; (b) the address of the PTW which holds the address of the selected page, in order to set a fault in it; and (c) a place to put the page in secondary memory.

The selection algorithm is based upon page usage. It is a particularly easy-to-implement version [4] of the "least-recently-used" algorithm [1, 8]. The hardware provides valuable assistance by, each time a page is referenced, setting ON a bit, called the *used bit*, in the corresponding PTW. The selection algorithm will not be described in detail here. However, it should be noted that candidates for removal are those pages described in the core map used list; therefore, each core map entry which appears in the used list must contain a pointer to the associated PTW (Figure 5) in order to permit examination of the used bit. The action of storing the PTW pointer in the core map entry must be added to the list of actions taken by the page fault handler when a page is moved into core (Section 8.4.).

Once the supervisor has selected the page to be removed, it takes the following steps:

Set the missing page switch ON in the PTW.

If no secondary memory has been assigned yet for this page, i.e. the segment map entry for this page holds a "null" address, assign a block of secondary memory and store its address in the segment map entry.

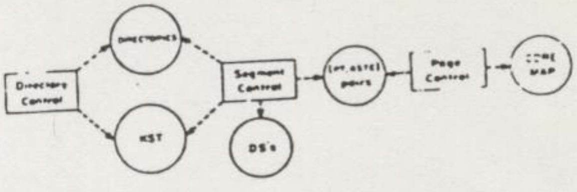
Issue an I/O request to move the page to secondary storage.

Upon completion of the I/O request, move the core map entry describing the freed block of core from the core map used list to the core map free list. This may be done in another process upon noticing the completion of the I/O request.

8.6 (PT, ASTE) Multiplexing

Core blocks can be multiplexed only among pages of active segments. The number of concurrently active segments is limited to the number of (PT, ASTE) pairs, which is, by far, smaller than the total number of segments in the virtual memory. Therefore (PT, ASTE) pairs must be multiplexed among all segments in the virtual memory.

Fig. 6. Supervisor functional modules and data bases.



When segment activation was described, a (PT, ASTE) pair was assumed available for assignment. In fact, this is not always the case. Making one segment active may imply making another segment inactive, thereby disassociating this other segment from its (PT, ASTE). Since all processes sharing the same segment will have the address of the PT in an SDW, it is essential to invalidate this address in all SDW's containing it before removing the page table.

This operation requires: (a) an algorithm to select a segment to be deactivated; (b) knowing all SDW's that contain the address of the page table of the selected segment, in order to invalidate this address; (c) moving the attributes contained in the ASTE back to the branch; and (d) changing the status of the segment from active to inactive in the branch.

The selection algorithm for deactivation, like the selection algorithm for page removal, is based on usage. When the last page of a segment is removed from core, the segment becomes a candidate for deactivation. The algorithm selects for deactivation the segment which has had no pages in core for the longest period of time, i.e. the segment which has been least recently used. Since the number of (PT, ASTE) pairs substantially exceeds the number of pageable blocks of core, it is always possible to find an active segment with no pages in core.

The ASTE must provide all the information needed for deactivating a segment. This means that during activation and connection, this information must be made available. During activation, a pointer to the branch must be placed in the ASTE; during connection, a pointer to the SDW must be placed in the ASTE. Since more than one SDW is connected to the same PT when the segment is shared by several processes, the supervisor must maintain a list of pointers to all connected SDW's. This list is called a connection list. See Figure 5.

After the selection algorithm chooses a (PT, ASTE) to be freed, the disassociation of the segment from its

PT, ASTE) is done in two steps: *disconnection* and *tivation.*

sconnection consists of storing a segment fault n SDW whose address appears in the connection in the ASTE. Deactivation consists of moving the ment map and the segment length from the ASTE back to the branch, resetting the active switch in the branch, and putting the (PT, ASTE) in the free list.

9. Structure of the Supervisor

Up to now supervisor functions have been described, but not the supervisor structure. In this section, the different components of the supervisor are presented d the ability of portions of the supervisor to utilize the virtual memory is discussed.

Functional Modules

Three functional modules can be identified in the supervisor described in Section 8; they are called *ectory control* (DC), *segment control* (SC), and *ge control* (PC).

DC performs all operations on segment attributes; it also maps pathnames into segment numbers in the T of the executing process. Data bases used by a process executing DC procedures are the directories and the KST of the process (Figure 6).

performs segment fault handling. Data bases used a process executing SC procedures are directories, the KST of the process, descriptor segments and (PT, STE) pairs.

PC performs page fault handling. Data bases used by a process executing PC procedures are (PT, ASTE) pairs and the core map.

.2 Use of PC in the Supervisor

One can observe that the page fault handler need not now if a missing page belongs to a user segment or to supervisor segment; it only expects to find the information it requires in the (PT, ASTE) of the segment o which the missing page belongs. Therefore, if all egments used in SC and DC are always active, then their pages need not be in core since PC can load them when they are referenced.

In order to make use of PC in the rest of the supervisor the following (temporary) assumption must be made.

Assumption 1

(a) All segments used in PC are always in core and are connected to the descriptor segment of each process.

h) All segments used in SC and DC are always active are connected to the descriptor segment of each ocess.

.3 Use of SC in the Supervisor

Assumption 1 is satisfactory in the Multics implementation *except for directories*.

The number of directory segments in the system may be very large and keeping them always active is not a realistic approach, since a large number of (PT, ASTE) pairs would have to be permanently assigned to them. It would be desirable to use SC to activate and connect directory segments only as needed.

A necessary condition for handling a segment fault for segment x in a process is that segment x be known to that process. Assuming that all directories are known to all processes, but not necessarily active, reference to a directory x may cause a segment fault. When handling this fault, the segment fault handler must reference the parent directory of segment x, where the branch for x is located. This reference to the parent of x could, in turn, cause a recursive invocation of the segment fault handler. These recursive invocations can propagate from directory to parent directory up to the root. If the root directory is always active and connected to each process, then the recursion is guaranteed to be finite and a segment fault for any directory can be handled.

The first assumption can be replaced by the following more satisfactory assumption (again temporary).

Assumption 2

(a) All segments used in PC are always in core and are connected to the descriptor segment of each process.

(b) All nondirectory segments used in SC and DC are always active and are connected to the descriptor segment of each process.

(c) The root directory is always active and connected to each process.

(d) All directories are always known to each process.

9.4 Use of the Make Known Facility in the Supervisor

However, it is unsatisfactory to keep all directories known to all processes because of the space that would be required in each KST. It would be more attractive if a directory could be made known to a process only when needed by the process.

Making a segment x known implies searching for its pathname in the KST. If not found, the parent of x must first be made known and so on up to the root. If the root directory is always known to all processes, then any directory can be made known to a process by calling recursively the Make Known facility of the supervisor.

Assumption 2 will now be replaced by the final assumption:

Final Assumption

(a) All segments used in PC are always in core and are connected to the descriptor segment of each process.

(b) All nondirectory segments used in SC and DC are always active and are connected to the descriptor segment of each process.

(c) The root directory is always active and connected to each process.

(d) The root directory is always known to each process.

Given the above assumption, supervisor segments, as well as user segments, can be stored in the virtual memory that the supervisor provides.

10. Summary

The most important points discussed in this paper are summarized below. They are grouped into two classes: the point of view of the user of the virtual memory, and the point of view of the supervisor itself.

User Point of View

The Multics virtual memory can contain a very large number of segments that are referenced by symbolic names.

Segment attributes are stored in special segments called directories, which are organized into a tree structure; by a naming convention known to the user, the symbolic name of a segment must be the pathname of the segment in the directory tree structure.

Any operation on directory segments must be done by calling the supervisor.

Any operation on a nondirectory segment can be done directly in accordance with the access rights that the user has for the segment; any word of any segment which resides in the virtual memory can be referenced with a pair (pathname, *i*) by the user.

Supervisor Point of View

The supervisor must simulate a large segmented memory which is directly addressable by symbolic name and such that any access to the memory is submitted to access rights checking.

The supervisor maintains a directory tree where it stores all segment attributes. It can retrieve the attributes of a segment, given the pathname of that segment.

The supervisor itself is organized into segments and runs in the address space of each user process.

Any segment, be it a directory or a nondirectory segment, is identified by its pathname but can be accessed only using a segment number. For each segment name the supervisor must assign a segment number by which the processor will address the segment in the process.

The processor accesses a word of a segment through the appropriate SDW and PTW, subject to the access rights recorded in the SDW.

A segment fault is generated by the processor whenever the page table address or access rights are missing in the SDW. The supervisor then, using the KST entry as a stepping stone, accesses the branch where it finds the needed information. If a PT is to be assigned, the supervisor may have to deactivate another segment.

A page fault is generated by the processor whenever a PTW does not contain a core address. The supervisor then, using the ASTE associated with the PT, moves the missing page from secondary storage to core. This may require the removal of another page.

Acknowledgments. This paper would be incomplete without acknowledgment of the people who worked so hard to build the virtual memory supervisor portion of Multics. Special mention goes to G.F. Clancy, M.R. Thompson, and S.H. Webber who, under the design leadership of R.C. Daley, have been involved in a major portion of the design and implementation effort. They were aided in earlier designs and implementations by C.A. Cushing, S.M. Jones, G.B. Krekeler, N.I. Morris, P.G. Neumann, R.K. Rathbun, J.D. Van Hausan, M.R. Wagner, and L.D. Whitehead. Recent implementations have also benefited from the contributions of S.D. Dunten and M.C. Turnquist. Contributions in the form of analyses and discussions have been made by F.J. Corbató, E.L. Glaser, J.H. Saltzer, and V.A. Vyssotsky.

Finally, our thanks go to P.A. Belmont, M.A. Meer, and D.L. Stone, who participated in studies leading to this formalized description of the Multics virtual memory.

Received April 1970; revised July 1971

References

1. Belady, L.A. A study of replacement algorithms for a virtual-storage computer. *IBM Systems J.* 5, 2 (1966), 78–101.
2. Comfort, W.T. A computing system design for user service. Proc. AFIPS 1965 FJCC, Vol. 27, Pt. 1, Spartan Books, New York, pp. 619–628.
3. Corbató, F.J., and Vyssotsky, V.A. Introduction and overview of the Multics system. Proc. AFIPS 1965 FJCC, Vol. 27, Pt. 1. Spartan Books, New York, pp. 185–196.
4. Corbató, F.J. A paging experiment with the Multics system. Included in a Festschrift published in honor of Prof. P.M. Morse. MIT Press, Cambridge, Mass., 1969.
5. Crisman, P.A. Ed. *The Compatible Time-Sharing System*: A Programmer's Guide, 2nd Ed., MIT Press, Cambridge, Mass., 1965.
6. Daley, R.C., and Neumann, P.G. A general-purpose file system for secondary storage. Proc. AFIPS 1965 FJCC, Vol. 27, Pt. 1. Spartan Books, New York, pp. 213–229.
7. Daley, R.C., and Dennis, J.B. Virtual memory, processes, and sharing in Multics. *Comm. ACM 11*, 5 (May 1968), 306–312.
8. Denning, P.J. The working set model for program behavior. *Comm. ACM 11*, 5 (May 1968), 323–333.
9. Denning, P. J. Virtual memory. *Computing Surveys 2*, 3 (Sept. 1970), 153–189.
10. Dennis, J.B. Segmentation and the design of multiprogrammed computer systems. *J.ACM 12*, 4 (Oct. 1965), 589–602.
11. Fotheringham, J. Dynamic storage allocation in the Atlas computer, including an automatic use of a backing store. *Comm. ACM 4*, 10 (Oct. 1961), 435–436.
12. Glaser, E.L., Couleur, J.F., and Oliver, G.A. System design of a computer for time sharing applications. Proc. AFIPS 1965, FJCC, Vol. 27, Pt. 1. Spartan Books, New York, pp. 197–202.
13. Graham, R.M. Protection in an information processing utility. *Comm. ACM 11*, 5 (May 1968), 365–369.
14. Saltzer, J. H. Traffic Control in a Multiplexed Computer System. Tech. Rep. No. MAC-TR-30 (Ph.D. Thesis), Project MAC, MIT, Cambridge, Mass., 1964.
15. The Descriptor – A definition of the B5000 Information Processing System. Burroughs Corp., Detroit, Mich., 1961.

FAULTS:

1) Micro-code builds a concealed stack frame "in" PCB

A) Concealed stack frame

Ø	RPH(seg#)	see fault
1	RPL (word #)	table #1
2	KEYS	see fault
3	F-Code	table #1
4	F-addr H	see fault
5	F-addr L	tabLe #1

B) Concealed stack is built at address next in PCB

2) Micro-code set RP to the fault vector in PCB plus fault offset. NOTE: Ring # is part of vector

3) Micro-code sets keys to 64V

4) Fetch next instruction

PASSES WHEN DATA NOT IN MEM.

SEG FAULT	(MICRO SECONDS)
PAGE FAULT	
STLB MISS	2 μSEC
CACHE MISS	.75 μSEC
DATA	80 NSEC

NAME	Fault Address PCB-Vector + Offset	F-code (16 Bits)	F-addr (32 Bits)	Ring	Saved RP	R-Mode Vector
Restrictor Instruction RXM	Vector of current ring + Ø	–	RP at time of fault	current	backed	'62
Process	FVO + '4	abort flags	RP at time of fault	Ø	current	'63
PAGE	PFV + '1Ø	–	RP at time of fault	Ø	backed	'64
SVC	Vector of current ring + '14	–	RP at time of fault	current	current	'65
UII	Vector of current ring + '2Ø	RPL	RP at time of fault	current	backed	'66
Illegal Instruction ILL	Vector of current ring + '4Ø	RPL	RP at time of fault	current	backed	'72
Acess Violation	FVØ + '44	'11	RP at time of fault	Ø	backed	'73
Arithmetic exception	Vector of current ring + '5Ø	'12	RP at time of fault	current	current	'74
Stack Overflow	FVØ + '54	'13	RP at time of fault	Ø	backed	'75
Segment	FVØ + '60	'14	RP at time of fault	Ø	backed	'76
Pointer	Vector of current ring + '64	'15	Address of pointer	current	backed	'77

TABLE

5) The first instruction of a fault handler is a CALF. A CALF instruction is the same as a PCL instruction except:

The stack frame built has additional information (see *)

CALF Stack Frame Header (V-Mode)

Ø	1	*	CALF set to 1 PCL set to Ø
1	STACK ROOT SEGMENT NUMBER		
2 3	RETURN POINTER	*	From concealed Stack
4 5	CALLER'S SAVED STACK BASE REGISTER		
6 7	CALLER' SAVED LINK BASE REGISTER		
8	CALLER'S SAVED KEYS	*	From concealed stack
9	LOCATION FOLLOWING CALL		
10	FAULT CODE	*	From concealed stack
11 12	FAULT ADDRESS	*	From concealed stack
13 14 15	RESERVED		

6) The CALF points to an ECB which describes the fault handler.

7) At this point the fault handler is entered and a return information is in the current stack. The fault handler is executed as a subroutine of the faulting routine.

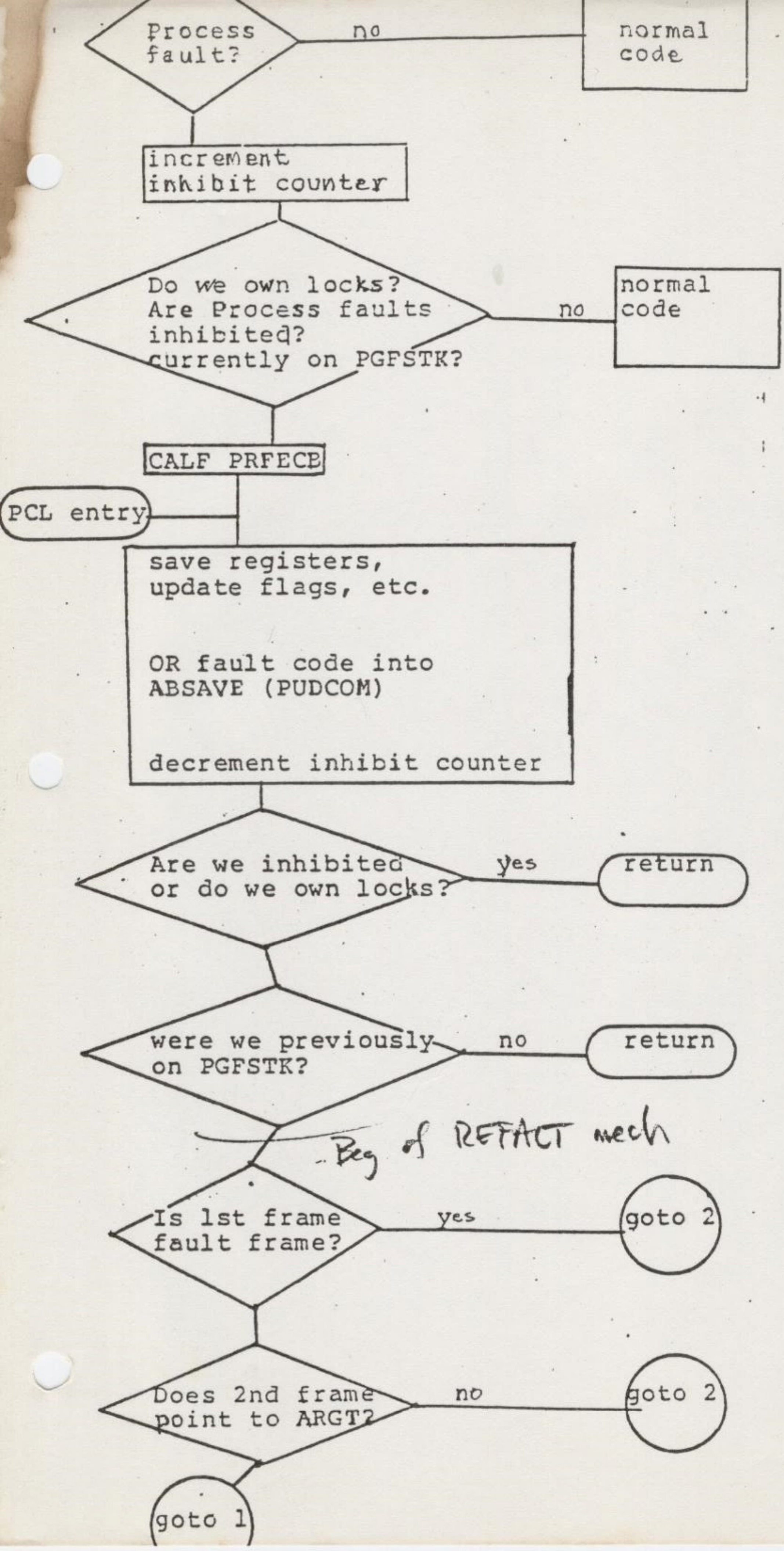
(Rev 18)

. REFAULT MECHANISM

- APPROXIMATELY 600 WORDS FOR STACK ONLY

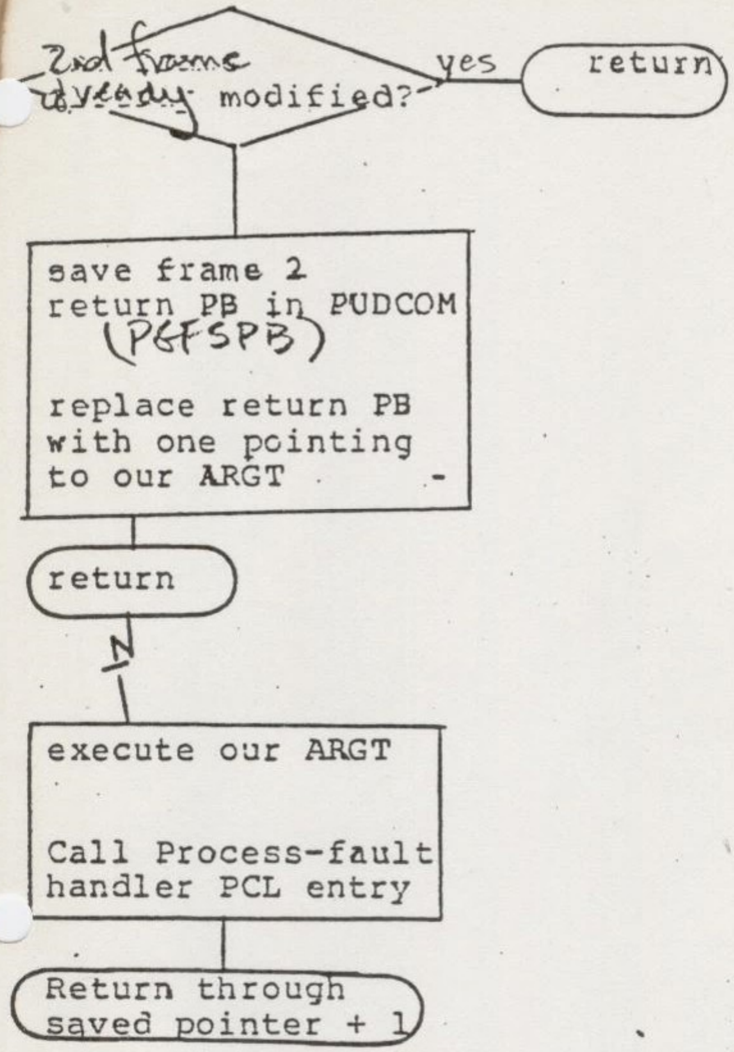
- MECHANISM FOR DEFERRING FAULTS UNTIL THE RETURN FROM PGFSTK

- REFALT MODIFIES THE RETURN PB IN A STACK FRAME AND PUSHES A FRAME IN THE CONCEALED STACK SO THAT A SIMULATED FAULT MAY BE TAKEN WHEN LEAVING PGFSTK

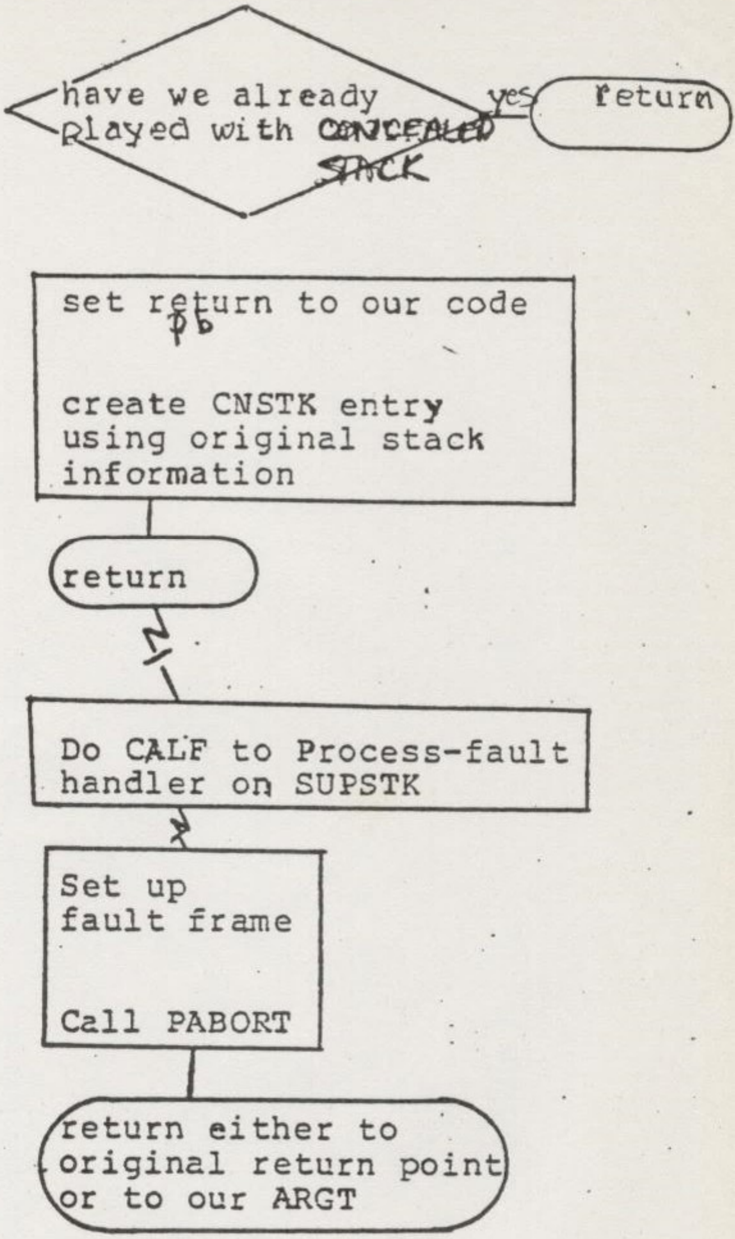

Process fault?
no
normal code
increment inhibit counter
Do we own locks? Are Process faults inhibited? currently on PGFSTK?
no
normal code
CALF PRFECB
PCL entry
save registers, update flags, etc.
OR fault code into ABSAVE (PUDCOM)
decrement inhibit counter
Are we inhibited or do we own locks?
yes
return
were we previously on PGFSTK?
no
return
Is 1st frame fault frame?
yes
goto 2
Does 2nd frame point to ARGT?
no
goto 2
goto 1

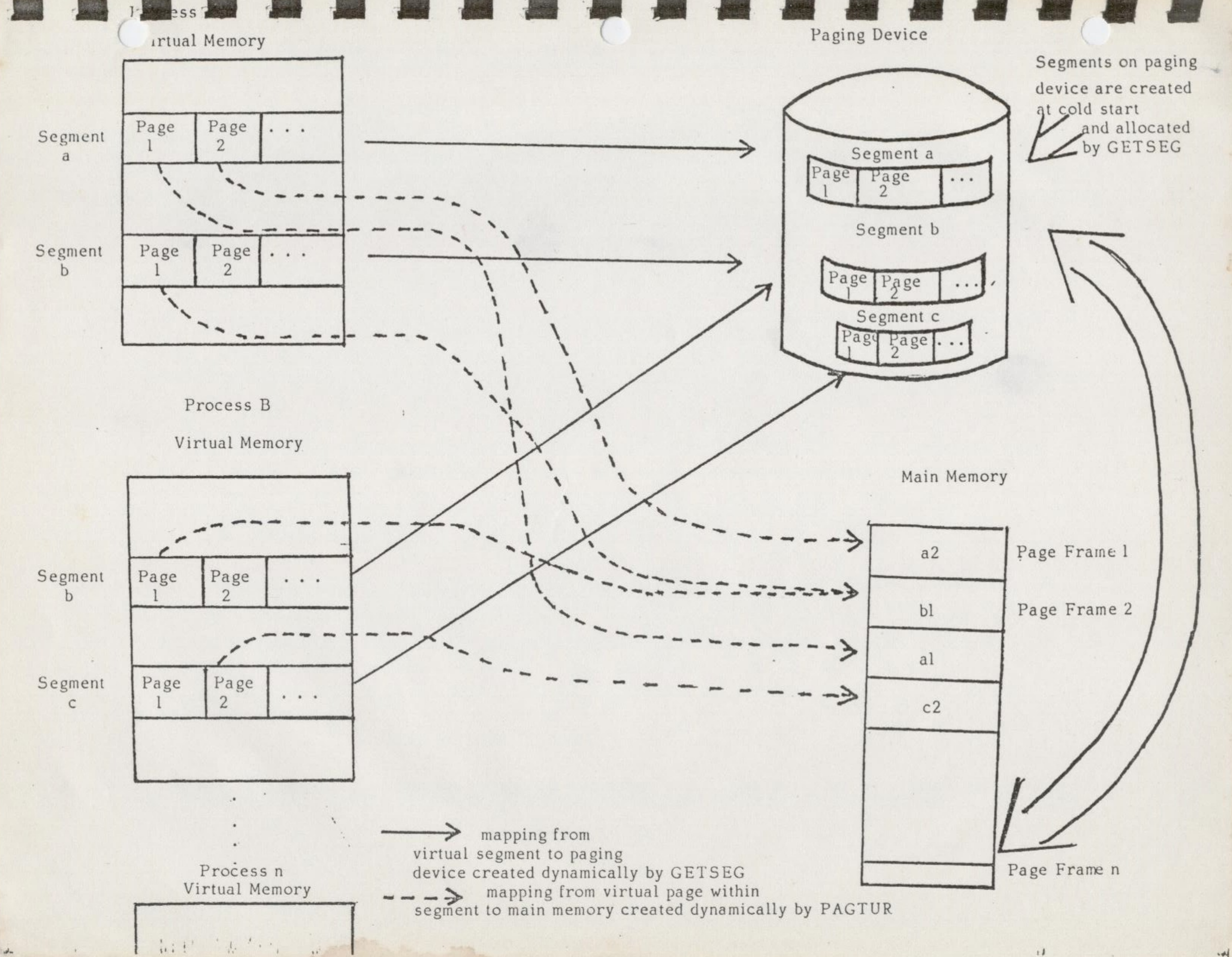
Beg of REFACT mech

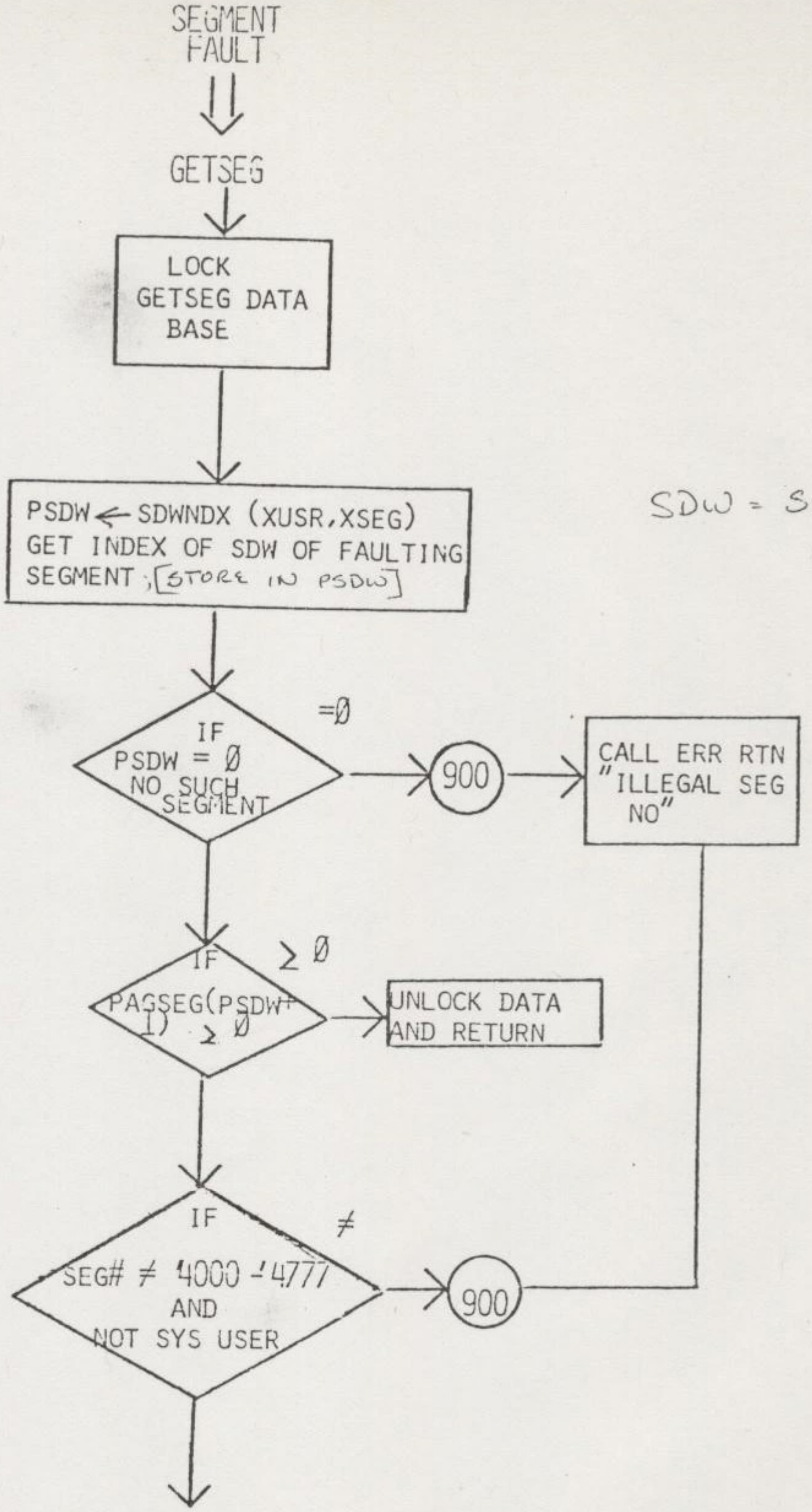
1


2nd frame ~~already~~ modified?
yes
return
save frame 2
return PB in PUDCOM
(PGFSPB)
replace return PB with one pointing to our ARGT
return
execute our ARGT
Call Process-fault handler PCL entry
Return through saved pointer + 1

2

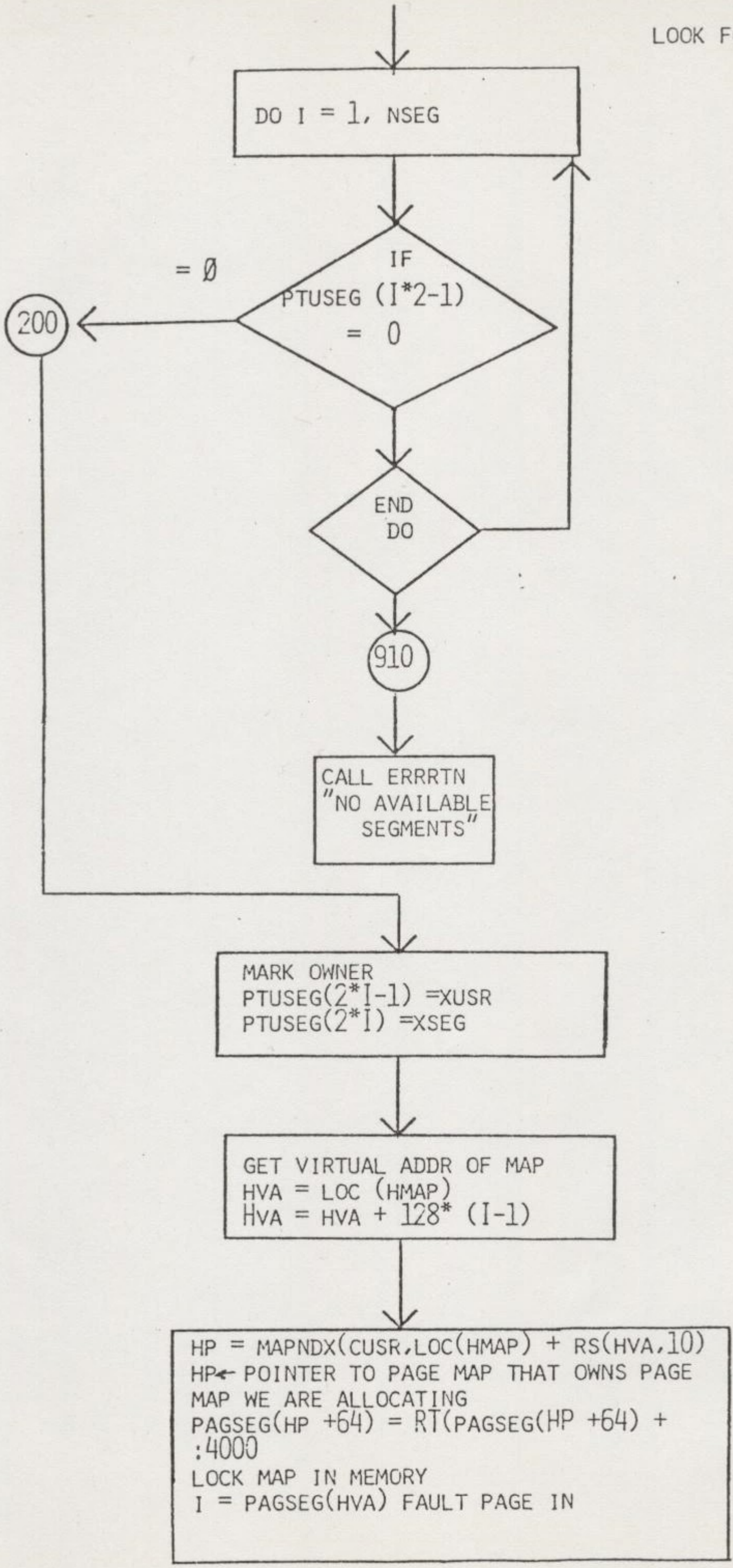

have we already played with CONCEALED STACK
yes
return
set return pb to our code
create CNSTK entry using original stack information
return
Do CALF to Process-fault handler on SUPSTK
Set up fault frame
Call PABORT
return either to original return point or to our ARGT

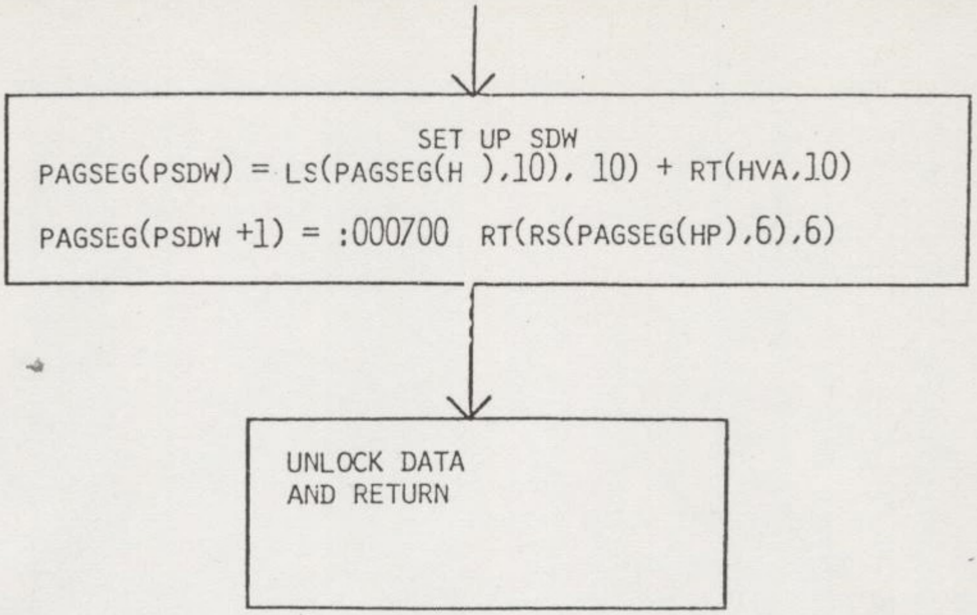

Process
Virtual Memory
Segment a
Page 1
Page 2
. . .
Segment b
Page 1
Page 2
. . .
Process B
Virtual Memory
Segment b
Page 1
Page 2
. . .
Segment c
Page 1
Page 2
. . .
Process n
Virtual Memory
Paging Device
Segment a
Page 1
Page 2
...
Segment b
Page 1
Page 2
...
Segment c
Page 1
Page 2
...
Segments on paging device are created at cold start and allocated by GETSEG
Main Memory
a2
bl
al
c2
Page Frame 1
Page Frame 2
Page Frame n
mapping from virtual segment to paging device created dynamically by GETSEG
mapping from virtual page within segment to main memory created dynamically by PAGTUR


SEGMENT FAULT
GETSEG
LOCK GETSEG DATA BASE
PSDW ← SDWNDX (XUSR, XSEG) GET INDEX OF SDW OF FAULTING SEGMENT ; [STORE IN PSDW]
IF PSDW = Ø NO SUCH SEGMENT
=Ø
900
CALL ERR RTN "ILLEGAL SEG NO"
IF PAGSEG(PSDW+1) ≥ Ø
≥ Ø
UNLOCK DATA AND RETURN
IF SEG# ≠ 4000 - 4777 AND NOT SYS USER
≠
900

SDW = SEG. DESCR. WORD

LOOK FOR A FREE PAGE MAP

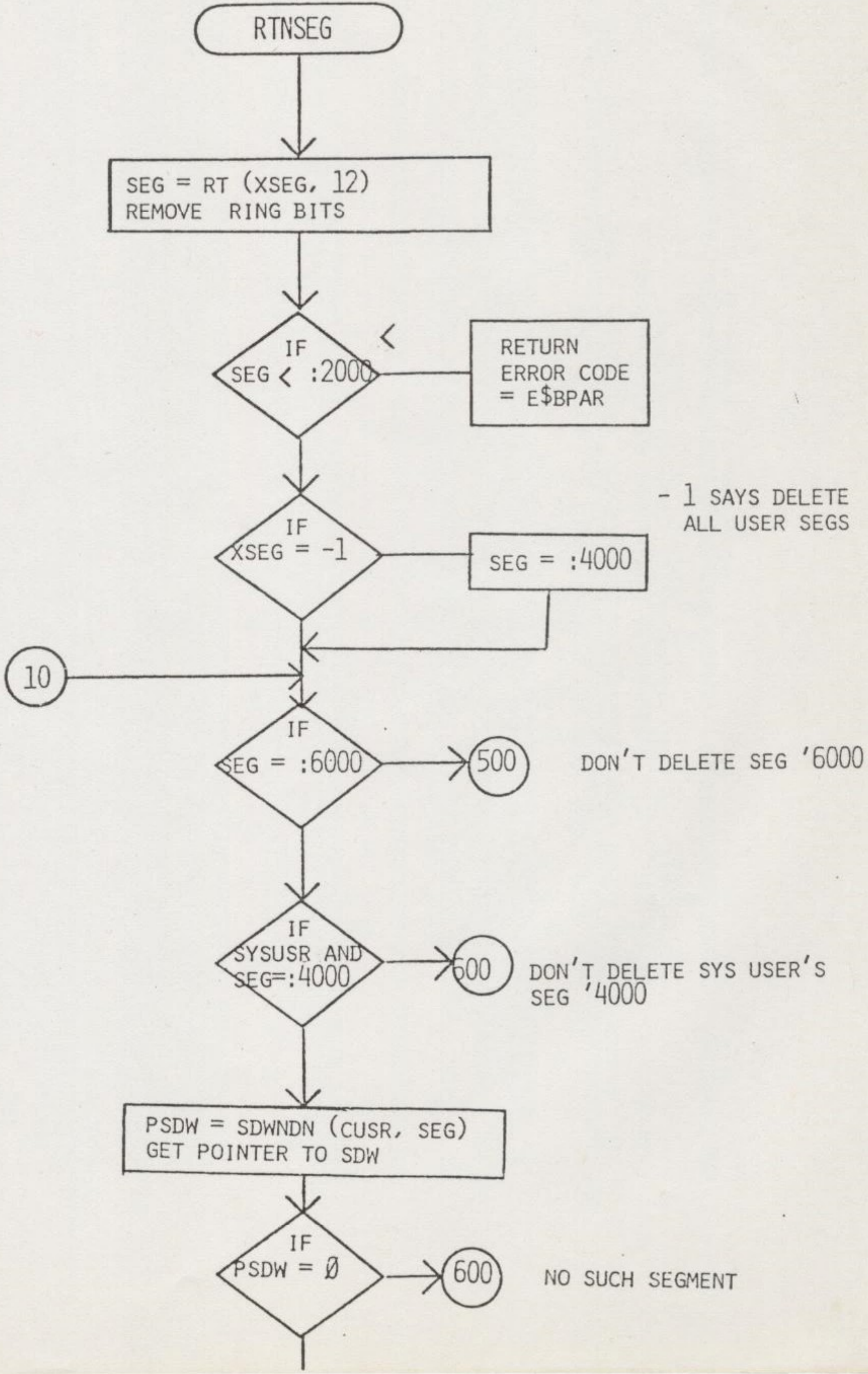


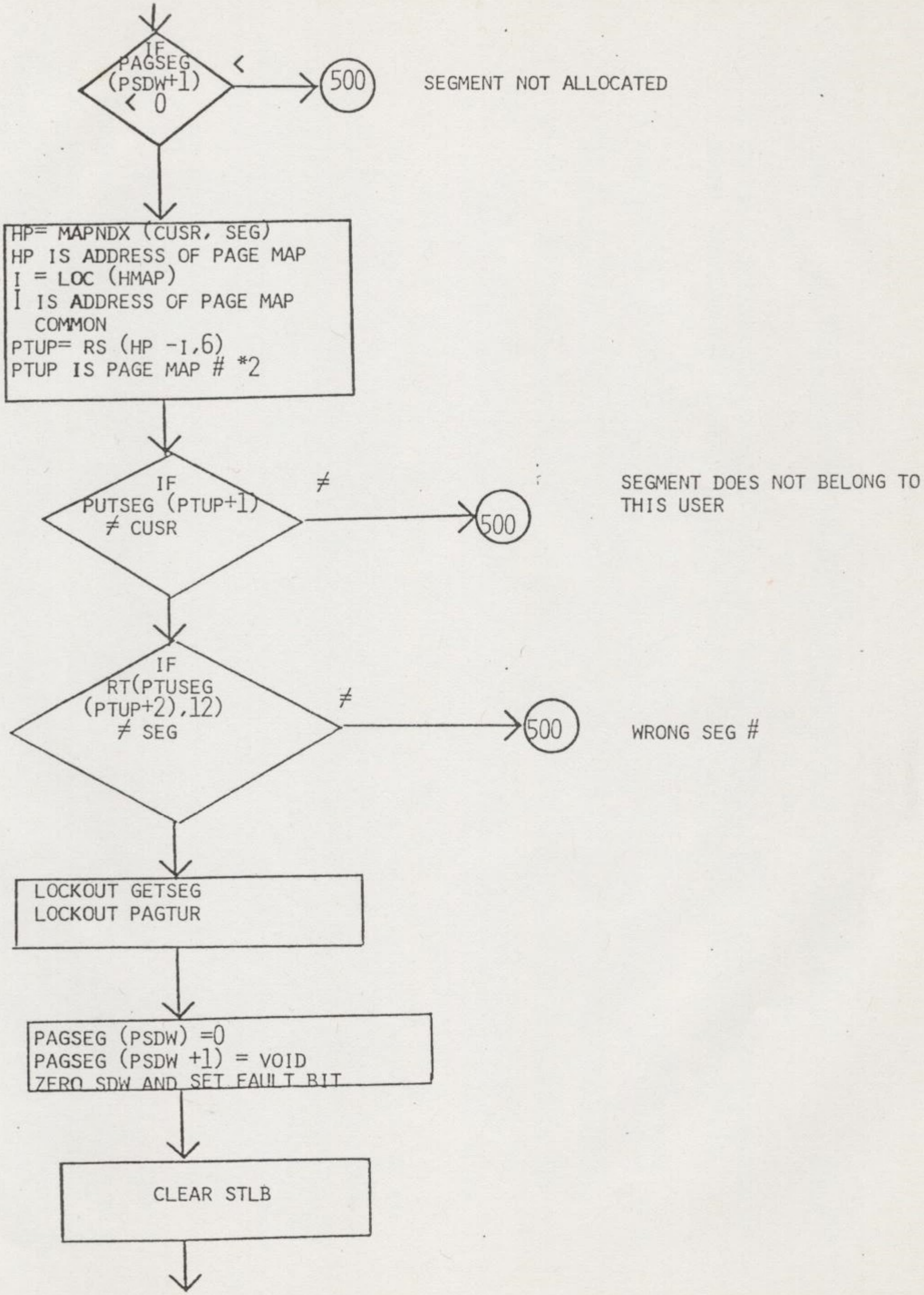

SET UP SDW
PAGSEG(PSDW) = LS(PAGSEG(H),10), 10) + RT(HVA,10)
PAGSEG(PSDW +1) = :000700 RT(RS(PAGSEG(HP),6),6)
UNLOCK DATA
AND RETURN

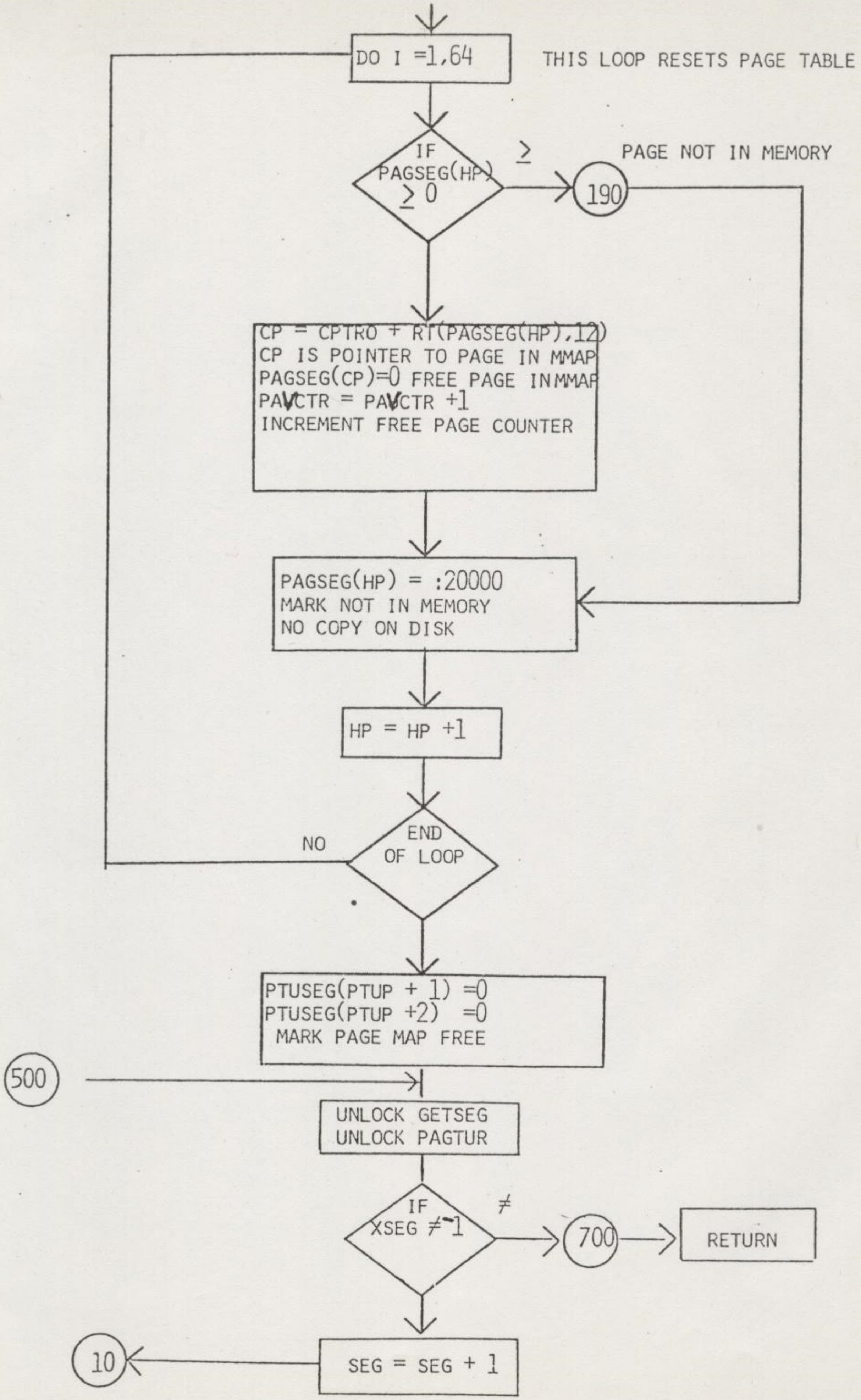
```
C      GETSEG, PRI400>KS, BLS, 03/04/78
C      ADD A SEGMENT TO A USER
C
       SUBROUTINE GETSEG(XUSR,XSEG)
       INTEGER XUSR,XSEG
C DVMCOM, PRI400>INSERT, JWP-BLS-JFC-REG-GMS-BIN-LDS-BEH-JCF-FVD, 12/02/78
       NOLIST
C      PUDC.F, PRI400, NIM, 04/02/78
       NOLIST
       SHORT CALL
     X      LOCKR, LOCKW, UNLKN, UNLKP,
     X      LOCKPS, LKPSW, UNLKPS, GUITON,
     X      FILPAG, INHBIT, ENABLE
C
       INTEGER I,PSDW,HVA,HP
C
C
       CALL LOCKW(SEGLCK)                              /* LOCK GETSEG DATA
       PSDW=SDWNDX(XUSR,XSEG)
       IF(PSDW.EQ.0) GOTO 900                          /* NO SUCH SEGMENT
       IF(PAGSEG(PSDW+1).GE.0) GOTO 250                /* SEG ALREADY EXISTS!
       IF(AND(XSEG,:4000).EQ.0 .AND. CUSR.NE.SUSR) GOTO 900
C
       DO 110 I=1,NSEG                                 /* LOOK FOR AVAILABLE PAGE-MAP
        IF(PTUSEG(2*I-1).EQ.0) GOTO 200
 110   CONTINUE
       GOTO 910                                        /* NONE AVAILABLE
C
C (FOUND AVAILABLE PAGE-MAP)
C
 200   PTUSEG(2*I-1)=XUSR                              /* MARK PAGE-MAP OWNED BY (XUSR
       PTUSEG(2*I)=XSEG
C
       HVA=LOC(HMAP)
       HVA=HVA+128*(I-1)                /* VA OF MAP
       HP=MAPNDX(CUSR,LOC(HMAP))+RS(HVA,10)            /* MAP PTR OF HVA
       PAGSEG(HP+64)=RT(PAGSEG(HP+64),14)+:40000       /* WIRE MAP
       I=PAGSEG(HVA)                            /* BRING MAP TO MEMORY
(0034)   C
(0035)         PAGSEG(PSDW)=LS(PAGSEG(HP),10)+RT(HVA,10)        /* SET SDW
(0036)         PAGSEG(PSDW+1)=:000700+RT(RS(PAGSEG(HP),6),6)  /* PLUS ACCESS CONTROLS
(0037)   290   CALL UNLKN(SEGLCK)
(0038)         RETURN
(0039)   C
(0040)   C (ERRORS)
(0041)   C
(0042)   900   CALL ERRRTN()SEG,0,'ILLEGAL SEGNO',13)
(0043)   910   CALL ERRRTN(0,0,'NO AVAILABLE SEGMENTS',21)
(0044)   C
(0045)         END
PROGRAM SIZE:   PROCEDURE - 000232     LINKAGE - 000067     STACK - 000034
0000 ERRORS [<GETSEG>FTN-REV16.2]
```

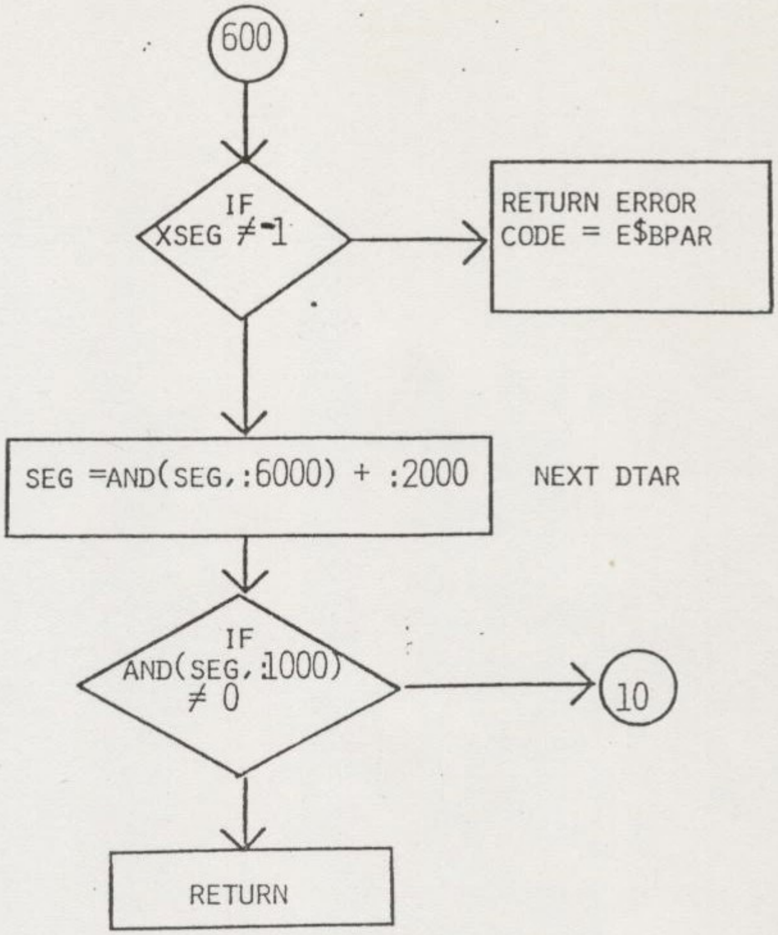
RTN SEG

SUBROUTINE TO FREE SEGMENTS CALLED BY DELSEG COMMAND AND LOGOUT




IF PAGSEG (PSDW+1) < 0
<
500
SEGMENT NOT ALLOCATED
HP= MAPNDX (CUSR, SEG)
HP IS ADDRESS OF PAGE MAP
I = LOC (HMAP)
I IS ADDRESS OF PAGE MAP COMMON
PTUP= RS (HP -I,6)
PTUP IS PAGE MAP # *2
IF PUTSEG (PTUP+1) ≠ CUSR
≠
500
SEGMENT DOES NOT BELONG TO THIS USER
IF RT(PTUSEG (PTUP+2),12) ≠ SEG
≠
500
WRONG SEG #
LOCKOUT GETSEG
LOCKOUT PAGTUR
PAGSEG (PSDW) =0
PAGSEG (PSDW +1) = VOID
ZERO SDW AND SET FAULT BIT
CLEAR STLB

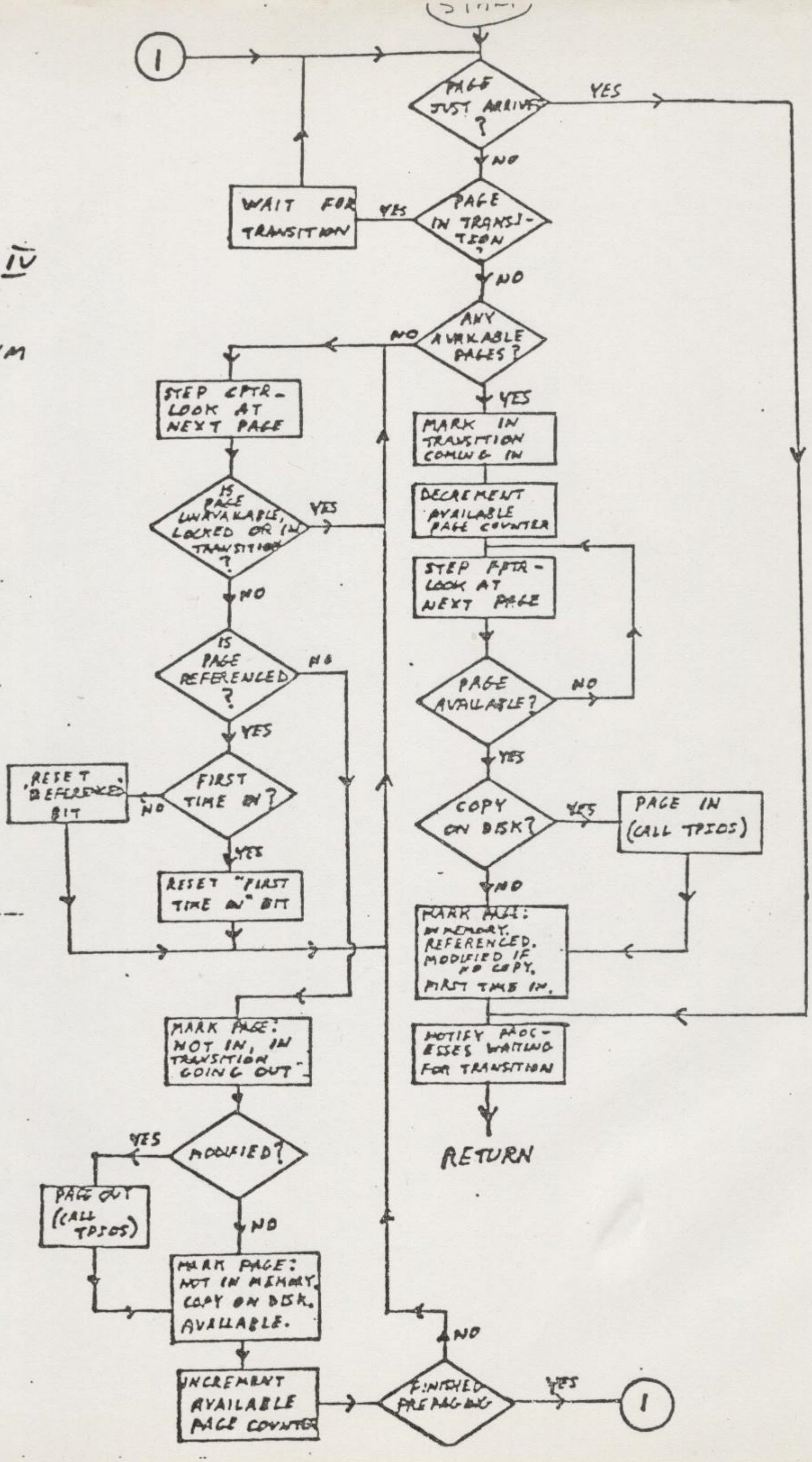

DO I =1,64
THIS LOOP RESETS PAGE TABLE
IF PAGSEG(HP) ≥ 0
≥
PAGE NOT IN MEMORY
190
CP = CPTRO + RT(PAGSEG(HP),12)
CP IS POINTER TO PAGE IN MMAP
PAGSEG(CP)=0 FREE PAGE INMMAP
PAVCTR = PAVCTR +1
INCREMENT FREE PAGE COUNTER
PAGSEG(HP) = :20000
MARK NOT IN MEMORY
NO COPY ON DISK
HP = HP +1
NO
END OF LOOP
PTUSEG(PTUP + 1) =0
PTUSEG(PTUP +2) =0
MARK PAGE MAP FREE
500
UNLOCK GETSEG
UNLOCK PAGTUR
IF XSEG ≠ -1
≠
700
RETURN
10
SEG = SEG + 1


600
IF XSEG ≠-1
RETURN ERROR CODE = E$BPAR
SEG =AND(SEG,:6000) + :2000
NEXT DTAR
IF AND(SEG,:1000) ≠ 0
10
RETURN

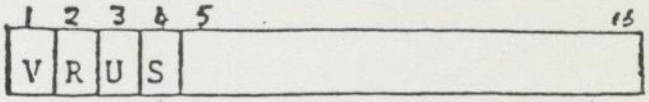
```
C      RTNSEG, FRI4CO>KS, ELS, 06/05/78
C      TO RETURN ONE SEGMENT OR ALL SEGS
       SUBROUTINE RTNSEG(XSEG,XCODE)
       INTEGER XSEG,XCODE
C DVYCOM, FRI400>INSERT, JWP-PLS-JPC-REG-GMS-BIN-LJS-BEH-JCF-FVD, 12/02/78
       NOLIST
C      FUDC.F, FRI4CO, NIM, 04/02/78
       NOLIST
C SYSCOM>ERRD.F       MNEMONIC CODES FOR FILE SYSTEM (FTN)     07/25/78
       NOLIST
       SHORT CALL
     X       LOCKR, LOCKW, UNLKN, UNLKF,
     X       LOCKFS, LKFSW, UNLKFS, QUITON,
     X       FILPAG, INHBIT, ENABLE
C
       INTEGER PSDW,HP,CP,I,SEG,PTUP
C
C (RELEASE PAGES)
C
       SEG=RT(XSEG,12)
       IF(SEG.LT.:2000) GOTO 500         /* NOBODY ALLOWED
       IF(XSEG.EQ.-1) SEG=:4000          /* IF LOOPING
C --- RETURN ONE SEGMENT (SEG)
10     IF(SEG.EQ.:6000) GOTO 500         /* SKIP STACK SEG
       IF(AND(SEG,:4000).EQ.0 .AND. CLSR.NE.SLSR) GOTO 600  /* NO ORDINARY LSR
       PSDW=SDWNDX(CLSR,SEG)
       IF(PSDW.EQ.0) GOTO 600                          /* NO SUCH SEGMENT
       IF(PAGSEG(PSDW+1).LT.0) GOTO 500              /* SEGMENT ALREADY MISSING
       HP=MAPNDX(CLSR,SEG)
       I=LOC(HMAP)
       PTUP=RS(HP-I,6)
       IF(PTUSEG(PTUP+1).NE.CLSR) GOTO 500
       IF(RT(PTUSEG(PTUP+2),12).NE.SEG) GOTO 500
C
       CALL LOCKW(SEGLCK)                            /* LOCK GETSEG DATA
       CALL LOCKW(PAGLCK)                            /* SHOULD NOT GET A PAGE-FAULT!
       PAGSEG(PSDW)=0
       PAGSEG(PSDW+1)=VOID                            /* SET FAULT BIT IN SDW
```

```
(0033)          CALL ITLBNZ                                        /* CLEAR WHOLE STLB
(0034)          DO 200 I=1,64
(0035)            IF(PAGSEG(HP).GE.0) GO TO 150                     /* PAGE NOT IN MEMORY
(0036)              CP=CPTRO+RT(PAGSEG(HP),12)                      /* PTR TO MMAP ENTRY FOR PAGE
(0037)              PAGSEG(CP)=0                                    /* MARK PAGE AVAILABLE
(0038)              PAVCTR=PAVCTR+1
(0039)  150         PAGSEG(HP)=:020000              /* MARK PAGE NOT IN MEMORY, NO COPY ON DISK
(0040)  200     HP=HP+1
(0041)  C
(0042)  C   (REMOVE FROM DESCRIPTOR TABLE)
(0043)          PTLSEG(PTLP+1)=0                                   /* PAGE-MAP AVAILABLE
(0044)          PTLSEG(PTLP+2)=0
(0045)  C
(0046)          CALL UNLKN(PAGLCK)
(0047)          CALL UNLKN(SEGLCK)
(0048)  C--- STEP TO NEXT SEG
(0049)  500     IF(XSEG.NE.-1) GOTO 700
(0050)          SEG=SEG+1
(0051)          GOTO 10
(0052)  C--- STEP TO NEXT DTAR
(0053)  600     IF(XSEG.NE.-1) GOTO 900
(0054)          SEG=AND(SEG,:6000)+:2000
(0055)          IF(AND(SEG,:10000).EQ.0) GOTO 10
(0056)  700     XCODE=0
(0057)          RETURN
(0058)  C
(0059)  900     XCODE=E$BPAR
(0060)          RETURN
(0061)          END
PROGRAM SIZE:    PROCEDURE - 000241     LINKAGE - 000070     STACK - 000034
0000 ERRORS [<RTNSEG>FTN-REV16.2]
```

REV 14
PRIMOS IV
PAGING
ALGORITHM


START
1
PAGE JUST ARRIVE ?
YES
NO
PAGE IN TRANSITION ?
YES
WAIT FOR TRANSITION
NO
ANY AVAILABLE PAGES?
NO
YES
MARK IN TRANSITION COMING IN
DECREMENT AVAILABLE PAGE COUNTER
STEP FPTR - LOOK AT NEXT PAGE
PAGE AVAILABLE?
NO
YES
COPY ON DISK?
YES
PAGE IN (CALL TPIOS)
NO
MARK PAGE: IN MEMORY. REFERENCED. MODIFIED IF NO COPY. FIRST TIME IN.
NOTIFY PROCESSES WAITING FOR TRANSITION
RETURN
STEP CPTR - LOOK AT NEXT PAGE
IS PAGE UNAVAILABLE, LOCKED OR IN TRANSITION ?
YES
NO
IS PAGE REFERENCED ?
NO
YES
FIRST TIME ON?
NO
RESET "REFERENCED" BIT
YES
RESET "FIRST TIME ON" BIT
MARK PAGE: NOT IN, IN TRANSITION GOING OUT
MODIFIED?
YES
PAGE OUT (CALL TPIOS)
NO
MARK PAGE: NOT IN MEMORY. COPY ON DISK. AVAILABLE.
INCREMENT AVAILABLE PAGE COUNTER
FINISHED PREPAGING
NO
YES
1

HMAP ENTRY: 16 BITS
(REV 14)



BIT 1 (V): Valid bit, set when page is in memory.

BIT 2 (R): Referenced bit, set by hardware when page is referrenced.

BIT 3 (U): Unmodified bit, reset by hardware when page is modified.

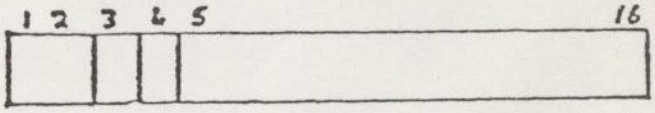
BIT 4 (S): Shared bit, set by software when memory page is shared by processors (inhibits cache)

BITS 5-16: High order 12 bits of physical page address (PPN), low order 10 bits taken as Ø.

If page not in memory, bits 3,5 define

00	not in, copy on disc
10	not in, no copy on disc
01	in transition, coming in
11	in transition, going out

LMAP ENTRY: 16 BITS
(MMAP +64) (REV 14)



BITS 1,2: Lock number (0 = not locked)

BIT 3: First-time bit

BIT 4: Use alternative paging device

BIT 5-16: Disc record index (for group of 8 pages)

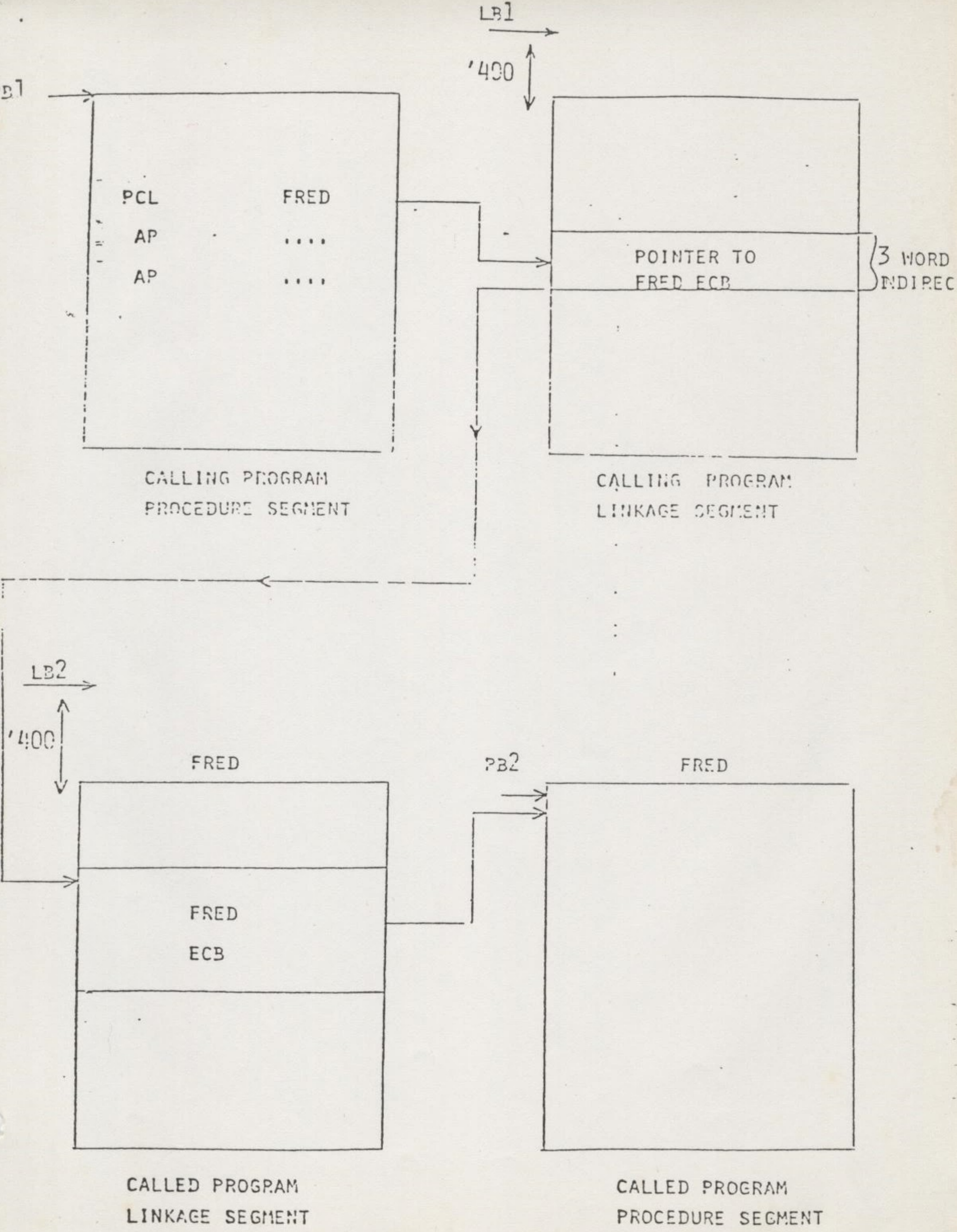
```
C        PAGTUR, PRMOS4, JWF-BLS-JPC-PVD-JCF-NIM, 12/15/78
C        PAGE TURNER.
C        PRIME COMPUTER, INC., SRCXXXX.C00
C        COPYRIGHT 1978, PRIME COMPUTER, INC., NATICK, MASS.
C
         SUBROUTINE PAGTUR(XPTR)
         INTEGER*4 XPTR
C
C        PAGTUR CONTAINS THE PAGE MANAGEMENT AND STRATEGY
C        FOR PRMOS4. PAGE-IN IS ON DEMAND. PAGE-OUT IS
C        ON AN APPROXIMATE LEAST-RECENTLY-USED ALGORITHM (WITH PRE-PAGING).
C        THE TWO PARTS OF THE PAGMAP (HMAP(64) AND LMAP(64)) HAVE ENTRIES DEFINED
C        AS FOLLOWS:
C
C        HMAP:      1     PAGE IN MEMORY (1 => TRUE)
C                   2     PAGE REFERENCED
C                   3     PAGE NOT MODIFIED
C                   4     PAGE SHARED
C                5-16     PHYSICAL PAGE NUMBER
C
C                IF PAGE NOT IN MEMORY BITS 3,5 DEFINE:
C
C                 00    NOT IN, COPY ON DISK
C                 10    NOT IN, NO COPY ON DISK
C                 01    IN TRANSITION, COMING IN
C                 11    IN TRANSITION, GOING OUT
C
C        LMAP:    1-2     LOCK NUMBER (0=UNLOCKED)
C                   3     FIRST-TIME (TO KEEP PAGE IN-MEMORY LONGER, AFTER PAGE-IN)
C                   4     USE ALTERNATE PAGING DISK
C                5-16     RECORD INDEX (ONE VALUE PER GROUP OF 8 PAGES)
C
C        MEMORY MAP TABLE FORMAT:
C            (1024 WORDS, ONE WORD PER REAL MEMORY PAGE)
C
C            IF <ENTRY> .NE. 0 PAGE IN USE. ENTRY IS PTR TO OWNER.
C            IF <ENTRY> .EQ. 0 PAGE AVAILABLE
C            IF <ENTRY> .EQ. -1 PAGE DOESN'T EXIST (MISSING MEMORY)
```

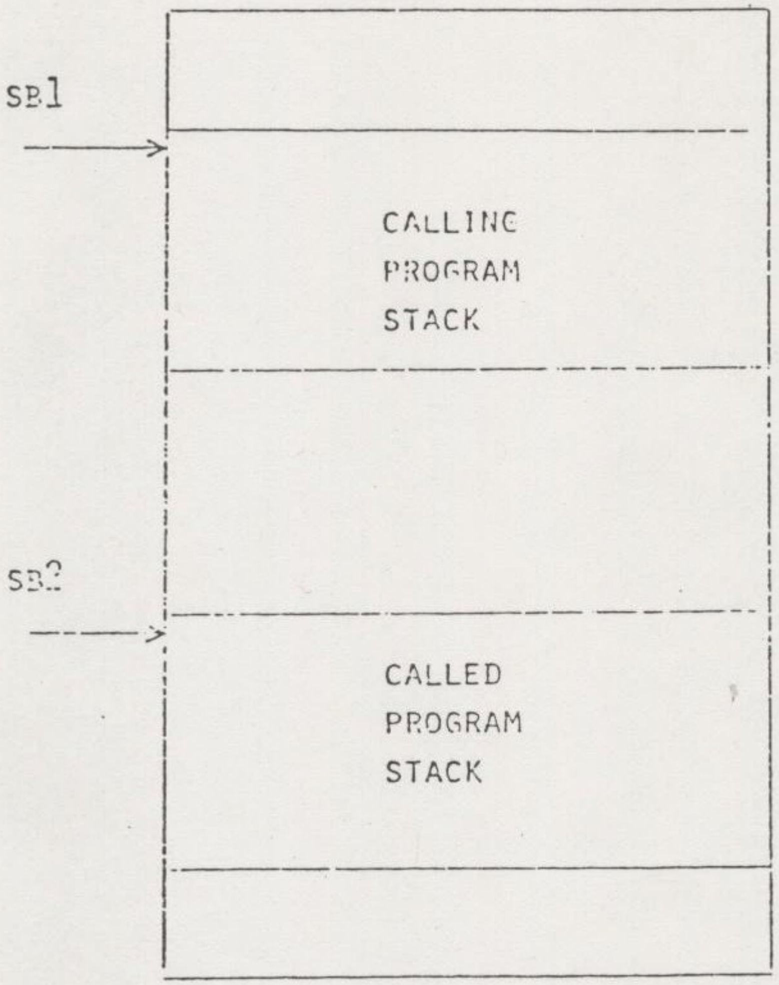
```
      C
      C
      C
      C DVMCOM, PRI400>INSERT, JWP-BLS-JPC-REG-GMS-BIN-LJS-BEH-JCF-FVD, 12/02/78
              NOLIST
      C       PUDC.F, PRI400, NIM, 04/02/78
              NOLIST
              SHORT CALL
             X      LOCKR, LOCKW, UNLKN, UNLKF,
             X      LOCKFS, LKFSW, UNLKFS, GLITCN,
             Y      FILPAG, INHBIT, ENABLE
      C
              COMMON /HMAPBF/ HMAPBF (64)             /* PAGE MAP FOR BUFSEG
              INTEGER HMAPBF
      C
      C       INTERNAL STORAGE AND VARIABLES.
              INTEGER HP,PMNT,PS,RA
              INTEGER*4 VPTR
      C
      100     CALL LOCKW(PAGLCK)                      /* LOCK PAGTUR DATA
      110     HP=MAPNDX(CUSR,XPTR)+RS(INTS(XPTR),10)/* PAGE-MAP ENTRY INDEX
              PMNT=PAGSEG(HP)                         /* SAVE PAGE-MAP ENTRY
              IF(PMNT.LT.0) GOTO 500                  /* PAGE JUST ARRIVED!
              IF(AND(PMNT,:4000).NE.0) GOTO 600       /* PAGE IN-TRANSITION
              IF(PAVCTR.EQ.0) GOTO 1000               /* NO AVAILABLE PAGES
              IF(RT(PAGSEG(HP+64),13).EQ.:17777) GOTO 2000      /* NOT ALLOCATED ON DISK
              PAGSEG(HP)=:4000                        /* MARK IN-TRANSITION, COMING-IN
      C (FIND FREE PAGE)
              PAVCTR=PAVCTR-1
      200     FPTR=FPTR+1                             /* STEP GLOBAL FREE-POINTER
              IF(FPTR.GE.CPTE) FPTR=CPTB
              IF(PAGSEG(FPTR).NE.0) GOTO 200          /* MMAP ENTRY: NOT AVAILABLE
              PAGSEG(FPTR)=HP                         /* MMAP: PAGE-OWNED BY ME
      C
              PS=FPTR-CPTB0                           /* COMPUTE PHYSICAL PAGE NUMBER
              RA=LS(PAGSEG(HP+64),3)+RT(HP,3)         /* DISK RECORD INDEX
              IF(AND(PMNT,:20000).NE.0) GOTO 225      /* BYPASS READ IF NO COPY-ON-DISK
                CALL UNLKN(PAGLCK)                    /* UNLOCK PAGTUR DATA
             VPTR = XPTR                               /* COPY POINTER
             IF (LT (HP, 10) .EQ. INTS (RT (LOC (HMAPEF), 16)))
        X       VPTR = LOC (REKDAT(1))                 /* USE WINDOW IF BUFSEG
             CALL TFICS(0,LT(VPTR,22),PS,RA,$900)   /* READ-IN PAGE
             CALL LOCKW(PAGLCK)
             PAGSEG(HP)=XOR(:160000,PS,AND(PMAT,:30000))/* IN MEM, USED,MOD IF NO-COP
      C                                                         PRESERVE SHARED
             GOTO 250
      C
      225   PAGSEG(HP)=XOR(:160000,PS,AND(PMAT,:30000))
            CALL FILPAG(XPTR)                          /* FILL PAGE WITH ZEROES
      250   CALL INHBIT                                /* GC PROTECTED FOR A MOMENT.
            PAGSEG(HP+64)=OR(PAGSEG(HP+64),:20000)/* SET FIRST-TIME BIT
            CALL ENABLE                                /* NOW WE'RE OKAY!
      C
      500   IF(PGNFYC.EQ.0) GOTO 550                   /* GLOBAL PAGE-NOTIFY COUNTER
            PGNFYC=PGNFYC-1
            CALL NOTIFY(1,PAGSEM)                      /* NOTIFY PROCESSES WAITING FOR TRAN
            GOTO 500
      C
      550   CALL UNLKN(PAGLCK)                         /* UNLOCK PAGTUR DATA
            RETURN
      C   (PAGE IN-TRANSITION)
      600   PGNFYC=PGNFYC+1                            /* INCREMENT GLOBAL PAGE-NOTIFY CTR.
            CALL UNLKN(PAGLCK)                         /* UNLOCK PAGTUR DATA
            CALL WAIT(PAGSEM)                          /* WAIT FOR ANY TRANSITION BIT.
            GOTO 100                                   /* AND TRY AGAIN
      C   (ERROR ON PAGE-IN)
      900   CALL LOCKW(PAGLCK)                         /* LOCK PAGTUR DATA
            IF(AND(INTS(RS(XPTR,16)),:6000).NE.:4000) GOTO 910/* IF SYSTEM PAGE
            PAGSEG(HP)=:140000+PS                      /* USER PAGE: GIVE IT TO HIM
            CALL FILPAG(XPTR)                          /* FILL PAGE WITH ZEROES
            GOTO 920
      910   PAGSEG(HP)=0                               /* SYSTEM PAGE: RELEASE
            MMAP(PS+1)=0
            PAVCTR=PAVCTR+1
      920   IF(PGNFYC.EQ.0) GOTO 930
            PGNFYC=PGNFYC-1                            /* AFTER NOTIFYING OTHERS
```

```
          CALL NOTIFY(1,PAGSEM)
          GOTO 920
    920   CALL ERRRTN(0,0,'PAGE-DSK',8)
    C
    C   (NO AVAILABLE PAGE)
    C
    1000  CALL ENABLE                                /* ALLOW INTERRUPTS BRIEFLY
          CPTR=CPTR+1                                /* STEP GLOBAL RELEASE PTR
          IF(CPTR.GE.CPTE) CPTR=CPTB
          HP=PAGSEG(CPTR)                            /* MMAP ENTRY
          IF(HP.EQ.0) GOTO 1000                      /* PAGE AVAILABLE
          IF(HP+1.EQ.0) GOTO 1000                    /* PAGE NOT AVAILABLE
          CALL INHBIT                                /* GC PROTECTED DURING CHECK.
          IF(LT(PAGSEG(HP+64),2).NE.0) GOTO 1000/* PAGE WIRED-DOWN
          PMNT=PAGSEG(HP)
          IF(PMNT.GE.0) GOTO 1000                    /* PAGE IN TRANSITION
          IF(AND(PMNT,:40000).EQ.0) GOTO 1010        /* NOT USED, TAKE IT
          IF(AND(PAGSEG(HP+64),:20000).NE.0) GOTO 1900    /* 1ST-TIME, CLEAR IT
           PAGSEG(HP)=PMNT-:40000                    /* CLEAR USED BIT, TRY NEXT TIME
           GOTO 1000
    C (FOUND PAGE: NOT 1ST BIT AND NOT USED)
    1010  PAGSEG(HP)=AND(PMNT,:13777)+:24000         /* MARK NOT-IN, IN TRANSITION GOING
          CALL ENABLE                                /* INTERRUPTS NOW OKAY
    C
          PS=CPTR-CPTRC                              /* PHYSICAL PAGE NUMBER
          CALL SPTLB(PS)                             /* FLUSH STLB
          RA=LS(PAGSEG(HP+64),3)+RT(HP,3)            /* RECORD INDEX
          IF(AND(PMNT,:20000).NE.0) GOTO 1020        /* BYPASS WRITE IF NOT MODIFIED
            CALL UNLKN(PAGLCK)                       /* UNLOCK PAGTUR DATA
            CALL TFICS(1,INTL(0),PS,RA,$1950)        /* WRITE-OUT PAGE
            CALL LOCKW(PAGLCK)                       /* LOCK PAGTUR DATA
    C
    1020  PAGSEG(HP)=AND(PMNT,:10000)                /* MARK NOT-IN, COPY ON DISK,
                                                        PRESERVED SHARED BIT.
    C
    1030  MMAP(PS+1)=0                               /* PAGE AVAILABLE
          PAVCTR=PAVCTR+1
          IF(PAVCTR.LT.PREPGK) GOTO 1000             /* CONTINUE PRE-PAGING
          GOTO 110                                   /* START ALL OVER
```

```
0148)  C (FIRST-TIME BIT ON)
0149)  1900  PAGSEG(HP+64)=PAGSEG(HP+64)-:20000      /* CLEAR 1ST-TIME BIT
0150)        GOTO 1000                               /* AND FIND ANOTHER PAGE
0151)  C  (ERROR ON WRITE)
0152)  1950  CALL LOCKW(PAGLCK)                      /* LOCK PAGTUR DATA
0153)        IF(MMAP(PS+1).EQ.-1) GOTO 1000          /* PAGE MAPPED OUT
0154)        PAGSEG(HP)=PMAT+:40000                  /* RESTORE AND MARK USED
0155)        GOTO 1000                               /* LEAVING PAGE FOR ANOTHER TRY
0156)  C  (NO DISK SPACE ALLOCATED FOR PAGE)
0157)  2000  ERRVEC(2)=XPTR                          /* ALTVAL(2)=WDNO
0158)        CALL ERRRTN(XPTR,0,'ILLEGAL PAGE REF.',17)  /* ALTVAL(1)=SEGNO
0159)        END
PROGRAM SIZE:   PROCEDURE - 000643    LINKAGE - 000132    STACK - 000032
0000 ERRORS [<PAGTUR>FTN-REV16.2]
```

PROCEDURE CALLS





STACK SEGMENT

ARGUMENT TEMPLATE

1 4	5	6	7 8	9	10	11 16
B	I	0	BASE REG	L	S	0————0
			W			

B = BIT NUMBER

I = INDIRECT BIT

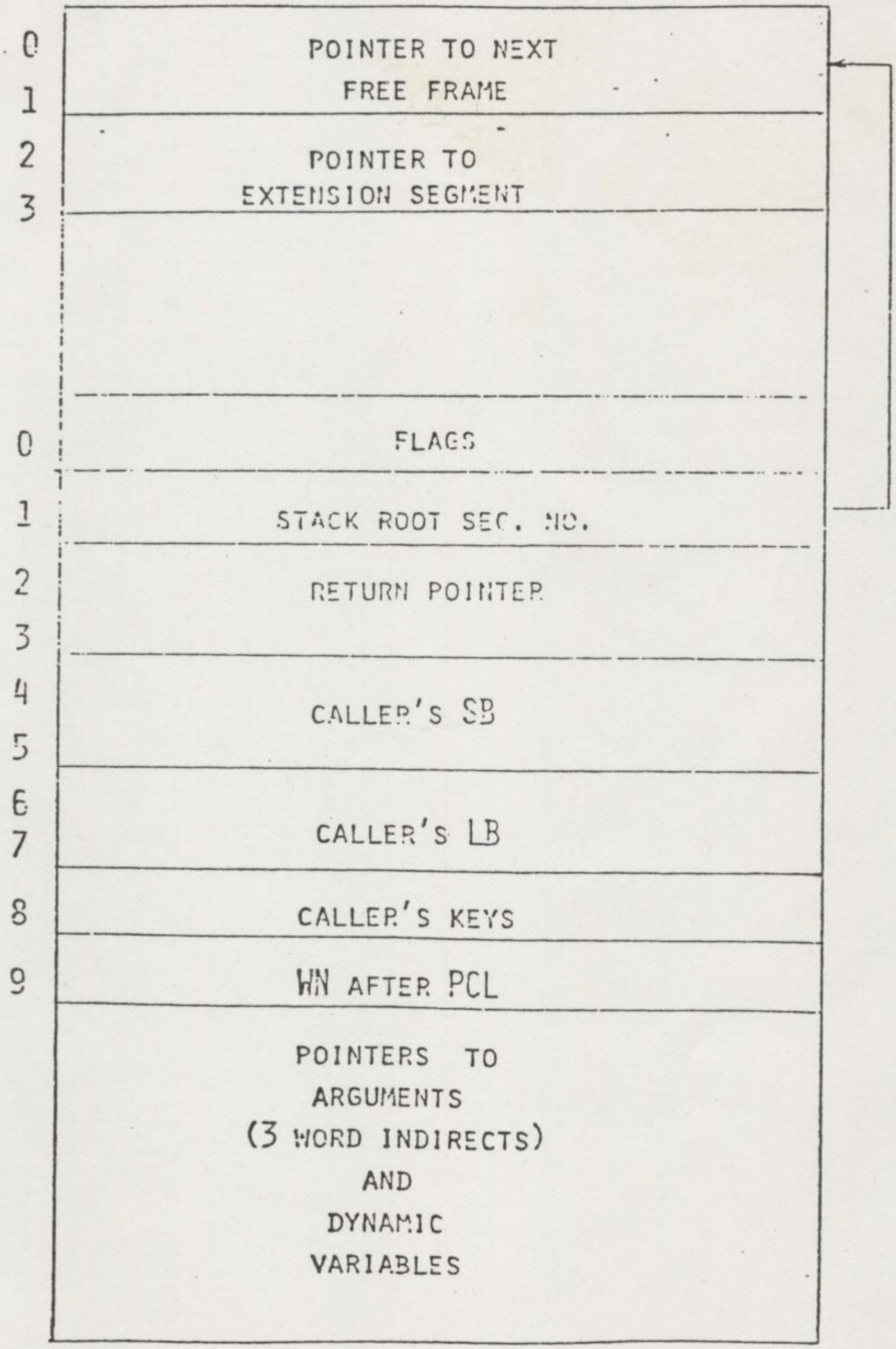
L = LAST BIT, LAST TEMPLATE FOR THIS PCL

S = STORE BIT, LAST TEMPLATE FOR THIS ARGUMENT

ENTRY CONTROL BLOCK (ECB)

0 1	POINTER TO FIRST EXECUTABLE STATEMENT OF CALLED PROGRAM
2	SIZE OF STACK FRAME
3	STACK ROOT SEG. NO.
4	ARG. DISPL.
5	NO. OF ARGS.
6 7	LINKAGE BASE OF CALLED PROGRAM
8	KEYS FOR CALLED PROGRAM
9 15	RESERVED MUST BE ZERO

STACK FRAME


0
1
POINTER TO NEXT FREE FRAME
2
3
POINTER TO EXTENSION SEGMENT
0
FLAGS
1
STACK ROOT SEC. NO.
2
3
RETURN POINTER
4
5
CALLER'S SB
6
7
CALLER'S LB
8
CALLER'S KEYS
9
WN AFTER PCL
POINTERS TO ARGUMENTS (3 WORD INDIRECTS) AND DYNAMIC VARIABLES

USE OF SUBROUTINES

(1) CALLING PROGRAM

CALL

- CALLS SUBROUTINE
- GENERATES PCL (PROCEDURE CALL)

PCL

- ADDRESSES AN ECB THROUGH A LINK
- CALCULATES RING NUMBER
- ALLOCATES STACK FRAME
- SAVES CALLER'S STATE
- INITIALISES STATE OF CALLED PROCEDURE
- TRANSFERS ARGUMENT POINTERS

AP

- GENERATES ARGUMENT POINTERS FOR PCL
- FOLLOWS PCL
- FORMAT

  AP ARG,TAG

  WHERE TAG MODIFIER CAN BE

S	VARIABLE IS ARGUMENT
SL	VARIABLE IS LAST ARGUMENT
*S	ARGUMENT IS INDIRECT
*SL	ARGUMENT IS INDIRECT AND LAST

2. SUBROUTINE

ARGT

- DOES LAST STEP OF PCL
- EXECUTED ONLY IF FAULT OCCURS DURING ARGUMENT TRANSFER
- MUST BE PRESENT IF ROUTINE REQUIRES ARGUMENTS

ECB

- GENERATES ENTRY CONTROL BLOCK (ECB) TO DEFINE A PROCEDURE ENTRY
- GOES INTO LINK FRAME
- FORMAT

```
LABEL ECB PFIRST,,ARGDISP,NARGS,
          SFSIZE,KEYS
```

WHERE:

PFIRST - POINTER TO FIRST EXECUTABLE STATEMENT

ARGDISP - DISPLACEMENT IN STACK FRAME OF ARGUMENT LIST (DEFAULT '12)

NARGS - NO. OF ARGUMENTS

SFSIZE - STACK FRAME SIZE, DEFAULT IS GIVEN BY DYNM

KEYS - KEYS, DEFAULT 64V

DYNM

- SPECIFIES VARIABLES TO GO INTO STACK FRAME
- EACH ARGUMENT REQUIRES 3 WORDS
- FORMAT

```
        DYNM  ARG(3),ARG2(3)
```

PRTN

- PROCEDURE RETURN
- RESTORES CALLER'S STATE
- DE-ALLOCATES STACK FRAME
- CALCULATES RING NUMBER

EXAMPLE

```
     SUPR    SUB,ECB
      ≀
SUB  ARGT              (ENTRY POINT)
     LDA     ARG1,* (GET FIRST ARG)
     STA     SUM
     LDA     ARG2,* (GET SECOND ARG)
     STA     COUNT
      ≀
     PRTN
     DYNM    ARG1(3),ARG2(3)
     DYNM    SUM,COUNT
      ≀
     LINK
ECB  ECB     SUB,,ARG1,2
      ≀
     END
```

NOTE

A MAINLINE PROGRAM IS EXECUTED USING THE PRIMOS IV SEG FACILITY.

TO ENABLE SEG TO ENTER THE PROGRAM THIS MUST INCLUDE AN ECB IN THE LINKAGE AREA.

THE END STATEMENT SHOULD BE FOLLOWED BY ,ADD WHERE ADD IS THE ADDRESS OF THE FIRST WORD OF THE ECB. THIS WILL ENABLE SEG TO SET UP THE ENTRY SEGMENT NUMBER AND WORD NUMBER.

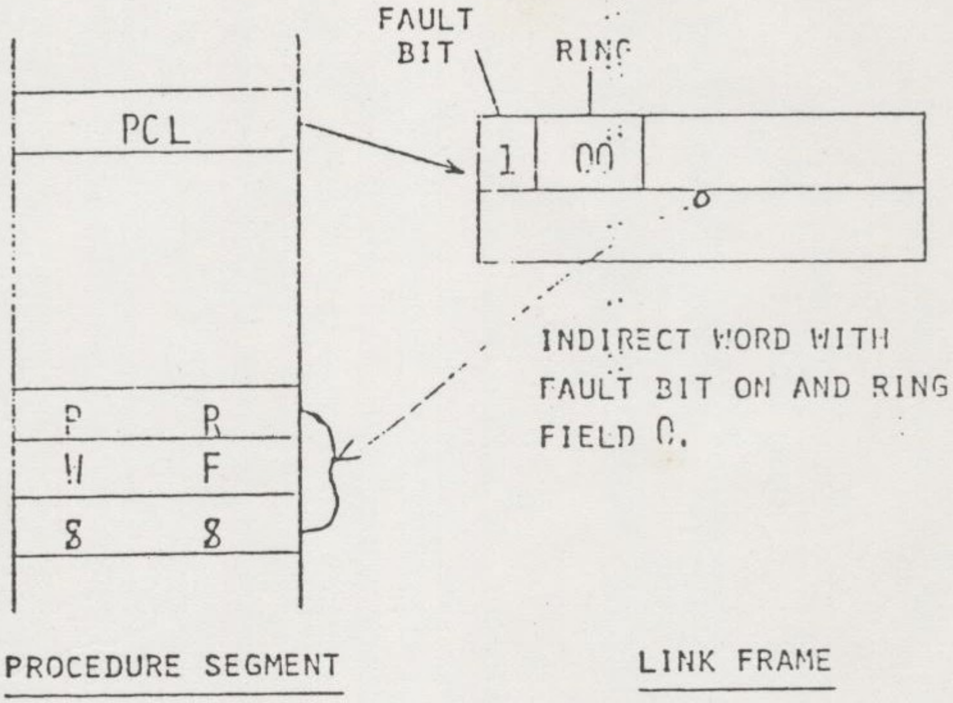
EXAMPLE

```
ADD   ....   FIRST EXECUTABLE INSTRUCTION



      LINK
ECB   ECB   ADD
      END,  ECB
```

DIRECT ENTRANCE CALLS

MANY PRIMOS IV ROUTINES, PREVIOUSLY REACHED BY SVC'S, ARE NOW REACHED (REV.14) BY DIRECT PROCEDURE CALL TO RING 0. THIS ELIMINATES THE OVERHEAD OF HANDLING THE SVC FAULT AND THE ATTENDANT ARGUMENT TRANSFER.

DIRECT ENTRANCE CALLS MAKE USE OF THE 'FAULT' BIT IN THE INDIRECT WORD.



THE ABOVE STRUCTURE IS CONSTRUCTED BY SEG WHEN IT ENCOUNTERS THE APPROPRIATE KIND OF ENTRY IN THE LIBRARY.

WHEN THE PCL IS EXECUTED AT RUN-TIME, THE FAULT BIT CAUSES A FAULT TO A ROUTINE WHICH FOLLOWS THE POINTER TO THE ASCII TEXT OF THE NAME.

DIRECT ENTRANCE CALLS

1) V-mode or I-mode entry to PRIMOS

2) Any service routines ring Ø

   a) I/O routines

   b) Access restricted data bases

3) D.E. call are entries for anyone into PRIMOS
   and the routine must protectect itself.

4) Dynamicly linked

CREATE DIRECT ENTRANCE·CALL

1) Put object code in Lib to tell seg this is a dynamicly linked routine.

```
        SEG
        DYNT      routine name
        END
```

2) Add a gate to Seg5 module of PRIMOS. Use gate Macro.

```
        GATE routine name  ,[PRIMOS name if diff]
```

   a) Note: Gate segment is search sequentially so order is important for efficiency.

   b) Note: adding gate may overflow the current size of Seg 5 and MAPGEN may need to be modified to increase the size of the segment.

3) Write the routine.

   a) Standard V-mode subroutine

   b) Must protect it's own entry point.

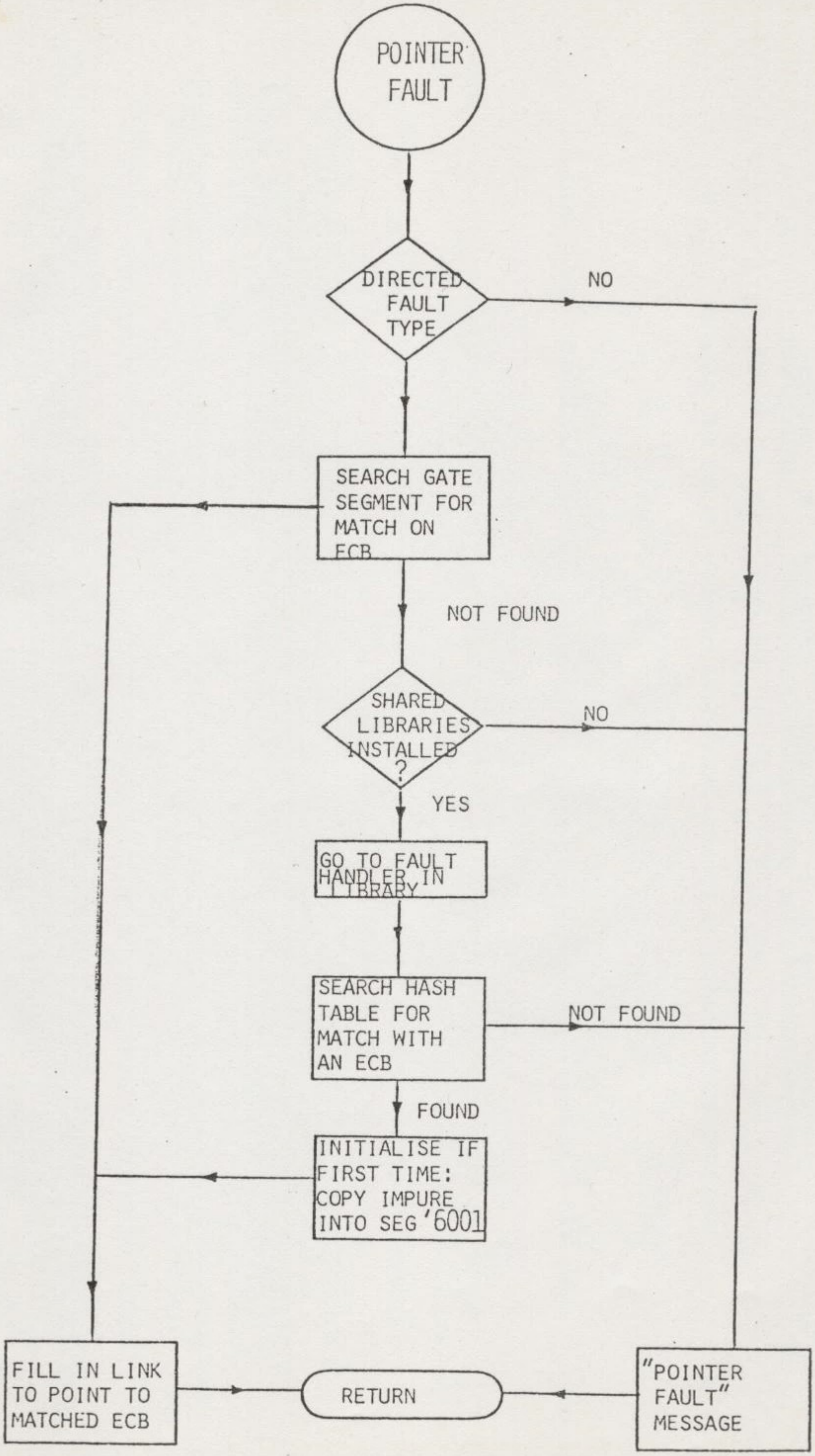
   c) Must validate all arguments

   d) Uses Ring Ø stack (seg #6000) set up by AINIT

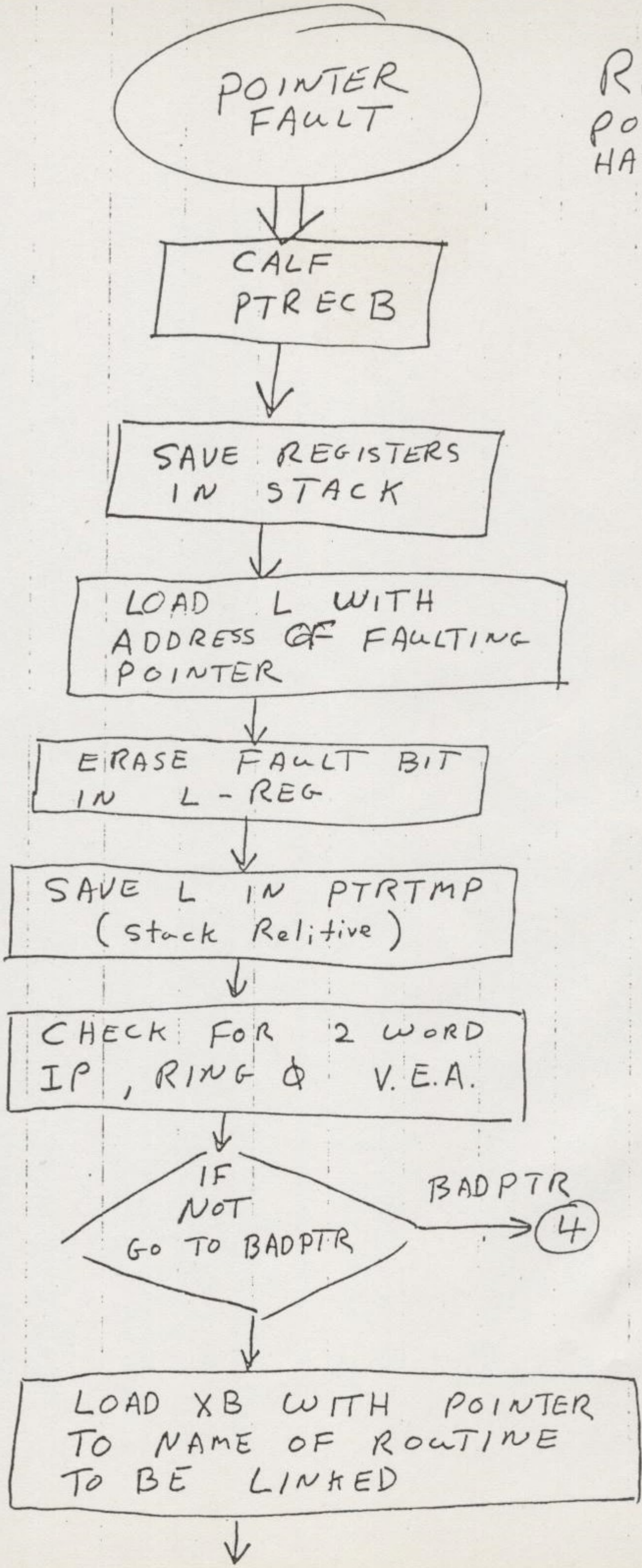
4) Load the routine with PRIMOS

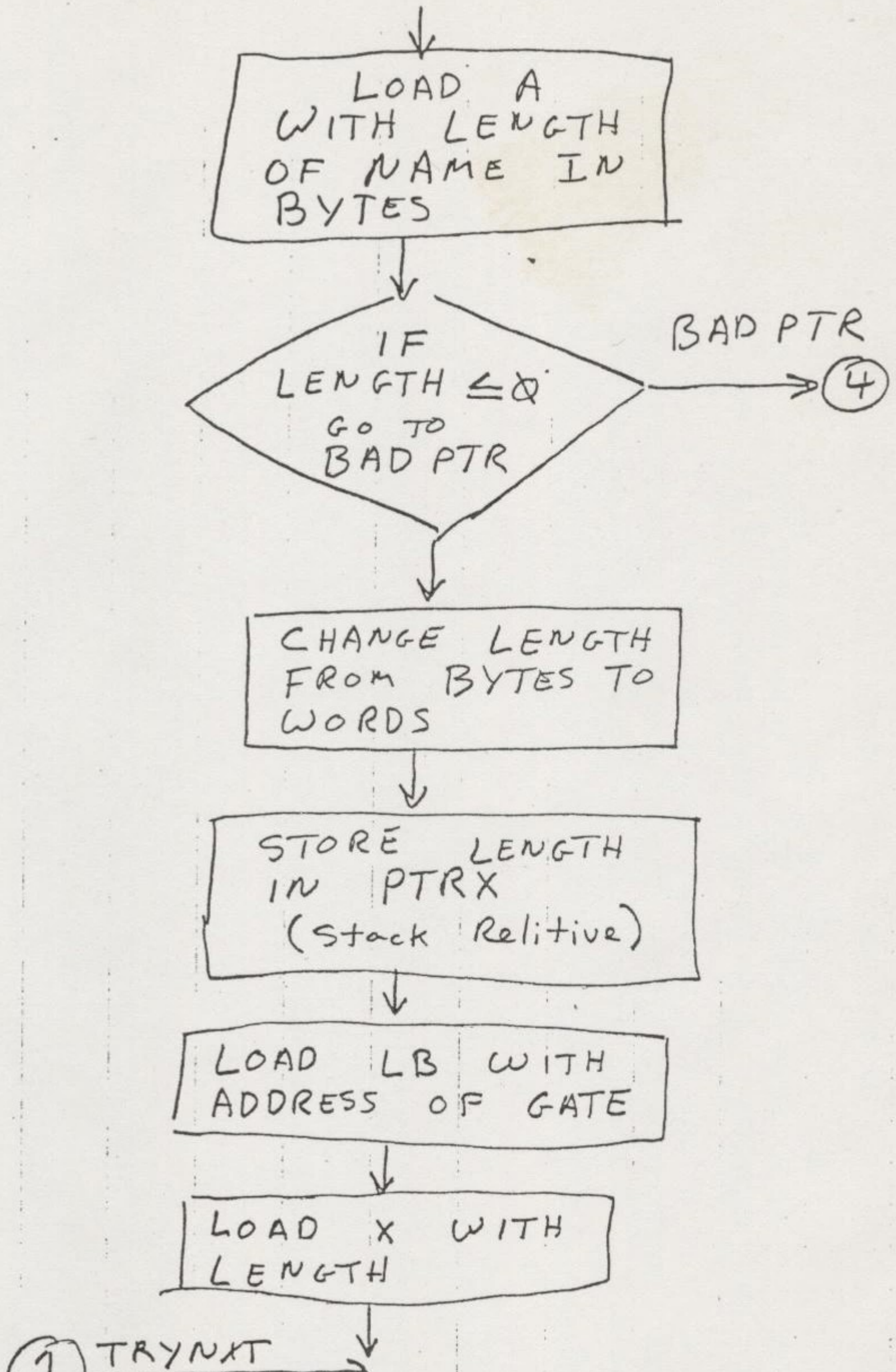
   a) May have to modify MAPGEN

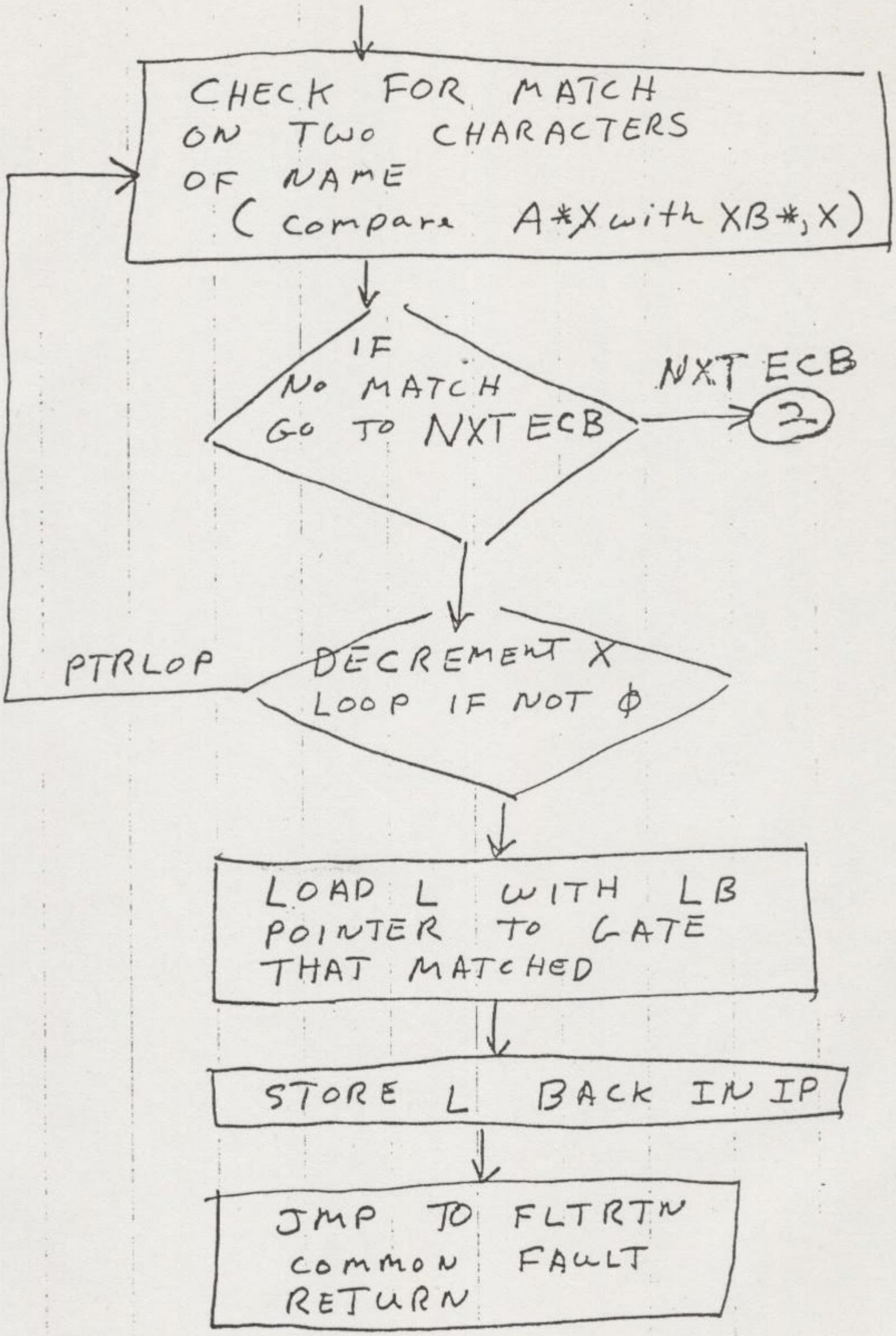
LINKING TO SHARED LIBRARIES (SIMPLIFIED)

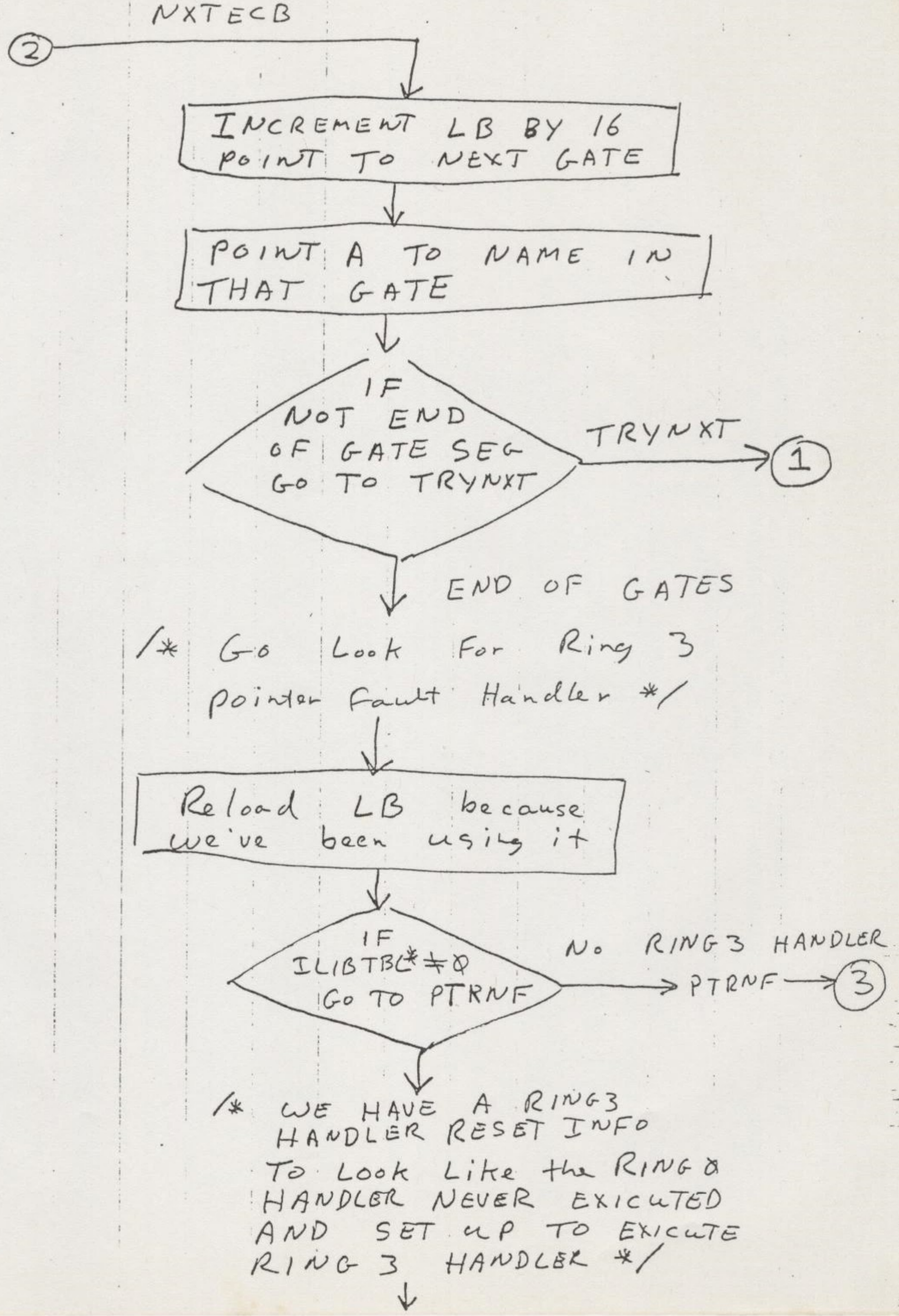

POINTER FAULT
DIRECTED FAULT TYPE
NO
SEARCH GATE SEGMENT FOR MATCH ON ECB
NOT FOUND
SHARED LIBRARIES INSTALLED ?
NO
YES
GO TO FAULT HANDLER IN LIBRARY
SEARCH HASH TABLE FOR MATCH WITH AN ECB
NOT FOUND
FOUND
INITIALISE IF FIRST TIME: COPY IMPURE INTO SEG '6001
FILL IN LINK TO POINT TO MATCHED ECB
RETURN
"POINTER FAULT" MESSAGE

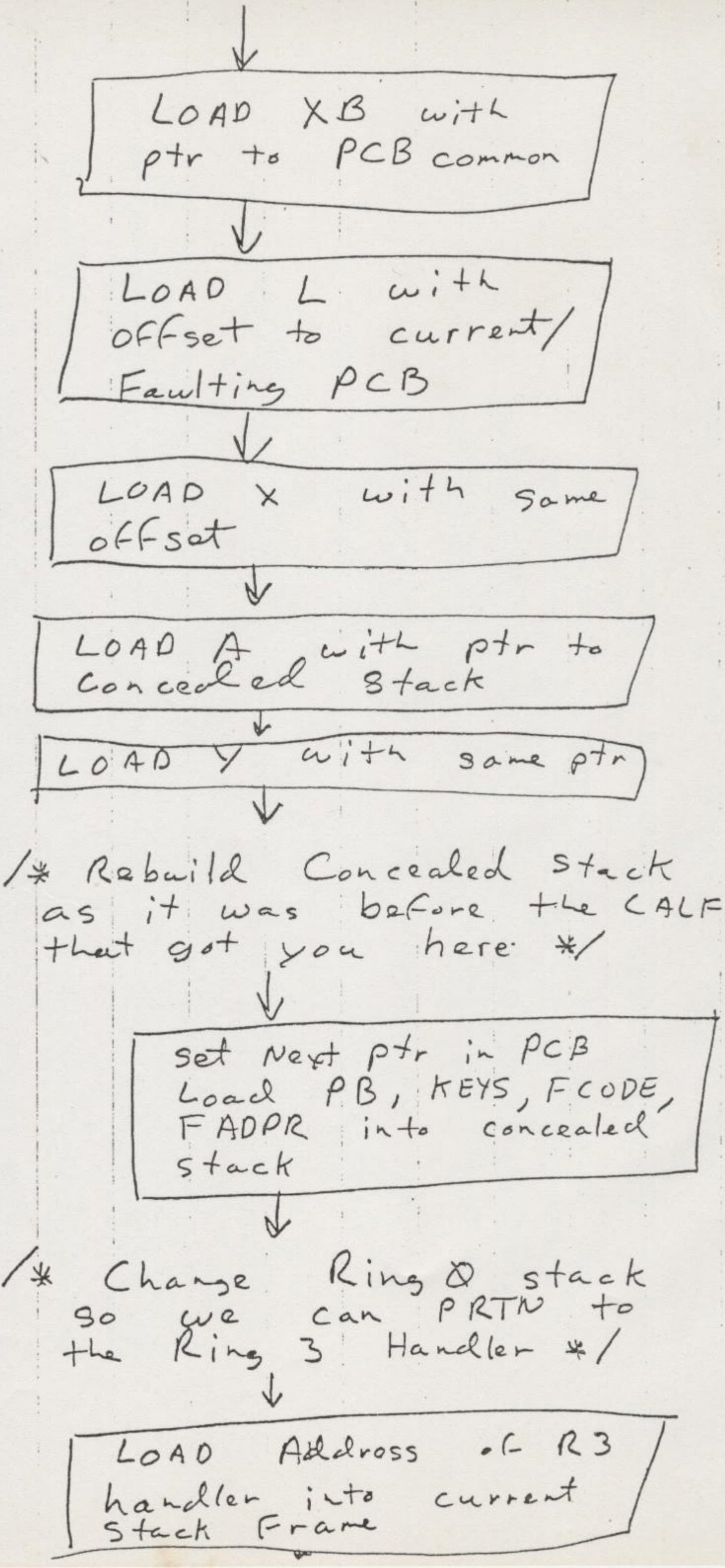
Ring Ø
Pointer Fault
HANDLER


POINTER FAULT
CALF PTRECB
SAVE REGISTERS IN STACK
LOAD L WITH ADDRESS OF FAULTING POINTER
ERASE FAULT BIT IN L-REG
SAVE L IN PTRTMP (stack Relitive)
CHECK FOR 2 WORD IP, RING Ø V.E.A.
IF NOT GO TO BADPTR
BADPTR
4
LOAD XB WITH POINTER TO NAME OF ROUTINE TO BE LINKED


LOAD A WITH LENGTH OF NAME IN BYTES
IF LENGTH ≤ Ø GO TO BAD PTR
BAD PTR
4
CHANGE LENGTH FROM BYTES TO WORDS
STORE LENGTH IN PTRX (Stack Relitive)
LOAD LB WITH ADDRESS OF GATE
LOAD X WITH LENGTH
1
TRYNXT
LOAD A WITH POINTER TO NAME IN GATE


CHECK FOR MATCH ON TWO CHARACTERS OF NAME
(compare A*X with XB*,X)
IF No MATCH GO TO NXTECB
NXTECB
2
PTRLOP
DECREMENT X LOOP IF NOT Ø
LOAD L WITH LB POINTER TO GATE THAT MATCHED
STORE L BACK IN IP
JMP TO FLTRTN COMMON FAULT RETURN


NXTECB
2
INCREMENT LB BY 16
POINT TO NEXT GATE
POINT A TO NAME IN THAT GATE
IF
NOT END
OF GATE SEG
GO TO TRYNXT
TRYNXT
1
END OF GATES
/* Go Look For Ring 3
pointer fault Handler */
Reload LB because
we've been using it
IF
ILIBTBC*≠Ø
GO TO PTRNF
NO RING3 HANDLER
PTRNF
3
/* WE HAVE A RING3
HANDLER RESET INFO
TO LOOK LIKE the RING Ø
HANDLER NEVER EXICUTED
AND SET UP TO EXICUTE
RING 3 HANDLER */



LOAD XB with ptr to PCB common

LOAD L with offset to current/Faulting PCB

LOAD X with same offset

LOAD A with ptr to Concealed Stack

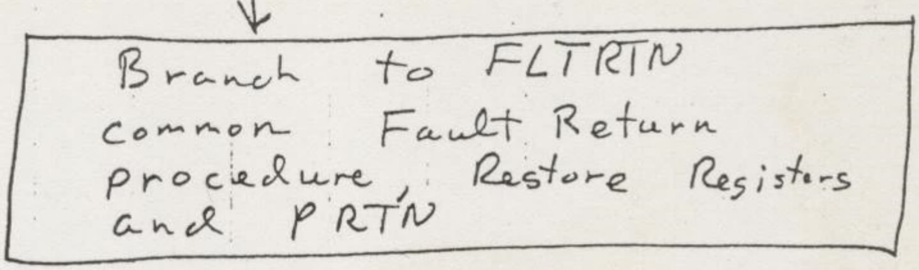
LOAD Y with same ptr

/* Rebuild Concealed stack
as it was before the CALF
that got you here */

set Next ptr in PCB
Load PB, KEYS, FCODE, FADPR into concealed stack

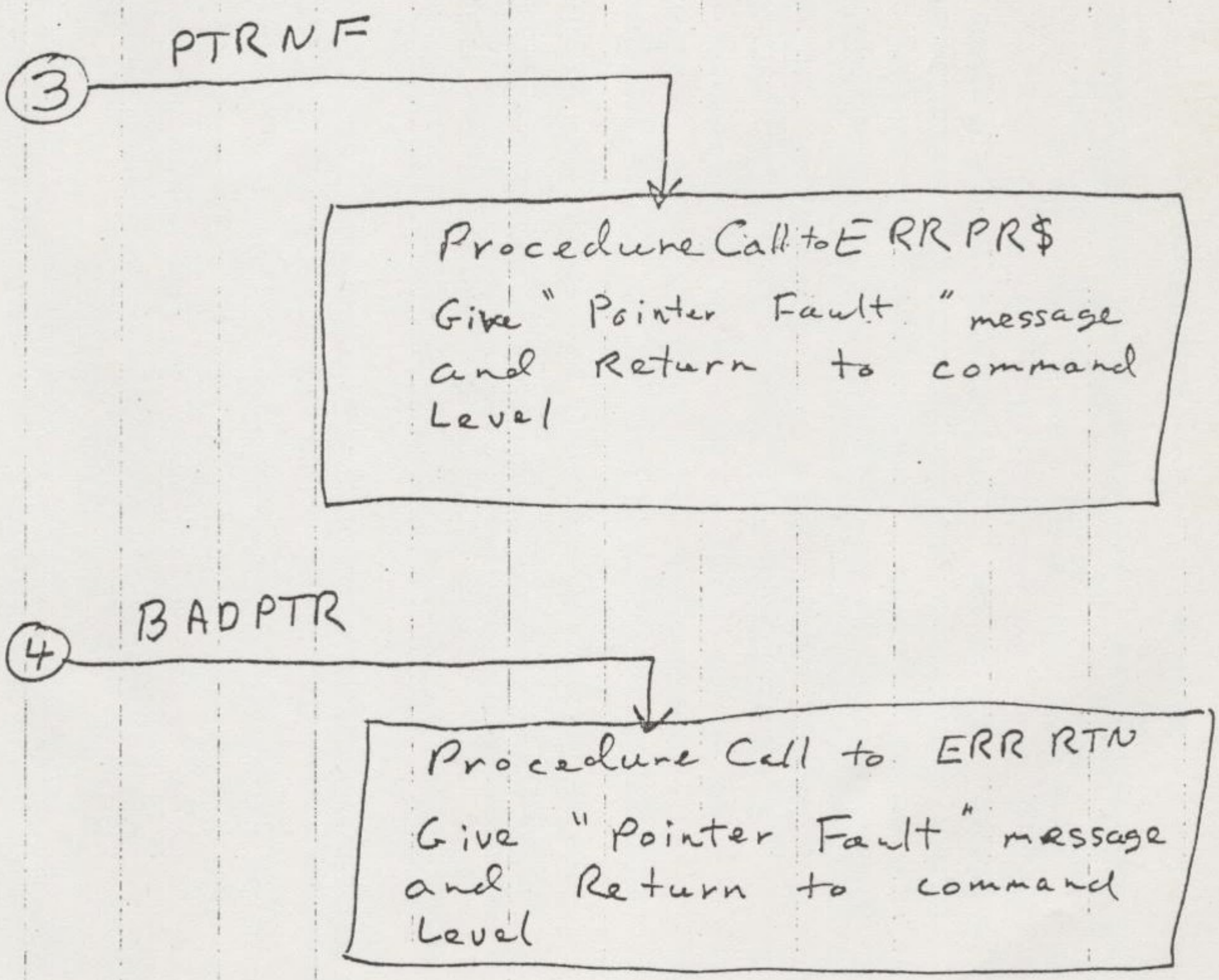
/* Change Ring 0 stack
so we can PRTN to
the Ring 3 Handler */

LOAD Address of R3 handler into current Stack Frame



Branch to FLTRTN common Fault Return procedure, Restore Registers and PRTN

/* NOTE because we changed the Ring Φ stack we go to the Ring 3 handler not back to the faulting Procedure and change the node of the machine to Ring3 */



Procedure Call to ERRPR$

Give " Pointer Fault " message and Return to command Level

Procedure Call to ERR RTN

Give "Pointer Fault" message and Return to command Level

```
                          (0646) *        POINTER-FAULT
                          (0647) *
                          (0648) PTRECB ECB    PTRF  ROOT= SUPSTK  FS= FLTFS+4
003363:        003374
               000020
               000011
               000000
               177400
               014000

003366:        006000
003365:        000054
               000050     (0649) PTRTMP EQU    SB%+FLTFS
               000052     (0650) PTRX   EQU    PTRTMP+2
                          (0651)
003374:        000715     (0652) PTRF   RSAV   RSAVE            SAVE USER STATE
003375: 000400.000015S
                          (0653)
003377: 045435.000013S (0654)         LDL    F_FADDR,*        PICK UP OFFENDING POINTER
003401:        140100  (0655)         SSP                     ERASE FAULT BIT
003402: 011415.000050S (0656)         STL    PTRTMP           SAVE POINTER
003404:     03.003610  (0657)         ANA    ='070000         TYPE=0?
003405: 140613.003545  (0658)         BNE    BADPTR           BRANCH IF NOT
003407: 065431.000050S (0659)         EAXB   PTRTMP,*         POINT TO LEN,NAME
003411: 005403.000000X (0660)         LDA    XB%              GET NAME LENGTH
003413: 140610.003545  (0661)         BLE    BADPTR           BRANCH IF .LE. 0
003415:        141206  (0662)         A1A                     LENGTH...
003416:        141050  (0663)         CAL
003417:        040477  (0664)         ARL    1
003420:     04.000052S (0665)         STA    PTRX             ...IN WORDS
003421: 067432.000444L (0666)         EALB   GATSG$
                       (0667)                                 START OF GATE SEGMENT
003423:     35.000052S (0668) TRYNXT LDX    PTRX
003424: 045402.000014L (0669) PTRLOP LDA    LB%+12,X          NAME LENGTH
003426: 053403.000000X (0670)         ERA    XB%,X
003430: 140613.003441  (0671)         BNE    NXTECB           BRANCH IF NO MATCH
003432: 140734.003424  (0672)         BDX    PTRLOP           TRY NEXT TWO CHARS OF NAMES
```

```
0034..: 003406.000000L (0673)        EAL   LB%              FULL MATCH -- GET ADDR OF ECB
003436: 051435.000013S (0674)        STL   F_FADDR,*        SNAP LINK
003440:       01.003326 (0675)       JMP   FLTRIN
                       (0676)
003441: 027412.000020L (0677) NXTECB EALB  LB%+16           NEXT ECB IN GATE SEGMENT
003443: 005402.000015L (0678)        LDA   LB%+13           CHECK FOR NAME
003445: 140613.003423  (0679)        BNE   TRYNXT           BRANCH IF IS ONE
003447: 067430.003371  (0680)        EALB  PTRECB+6,*       RELOAD LINK BASE
                       (0681) * SETUP FOR RING-3 FAULT-HANDLER
                       (0682) *
003451: 045420.003722  (0683)        LDA   ILIBTBL,*        SEE IF ANY R3 HANDLER
003453: 140612.003523  (0684)        BEQ   PTRNF            ENTRY NOT FOUND
003455: 065432.000446L (0685)        EAXB  PCBSEG           XB -> PCB SEGMENT
003457: 013404.000025P (0686)        LDLR  OWNER
003461:       35.000002A (0687)      LDX   2                X -> MY PCB
003462: 045403.000075X (0688)        LDA   XB%+PCSK+1,X     CONCEALED-STACK NEXT
003464:         140505 (0689)        TAY                    Y -> CURRENT 6WD ENTRY
003465: 063403.000076X (0690)        CAS   XB%+PCSK+2,X     CONCEALED-STACK LAST
003467:       01.003471 (0691)       JMP   *+2
003470:       01.003520 (0692)       JMP   PTRF3            EQUALS LAST, MUST RESET
003471:       06.003611 (0693)       ADD   =6
003472: 051403.000075X (0694) PTRF2  STA   XB%+PCSK+1,X     SET NEXT PTR
                       (0695) *
00347   005415.000002S (0696)        LDL   F_PB             MOVE INFO TO CONCEALED-STACK
003476: 011437.000000X (0697)        STL   XB%,Y
003500:       02.000012S (0698)      LDA   F_FCODE
003501:         140314 (0699)        TAB
003502:       02.000010S (0700)      LDA   F_KEYS
003503: 011437.000002X (0701)        STL   XB%+2,Y
003505: 005415.000013S (0702)        LDL   F_FADDR
003507: 011437.000004X (0703)        STL   XB%+4,Y
                       (0704) *
003511: 045434.003722  (0705)        LDL   ILIBTBL,*        R3 FAULT-HANDLER
003513: 011415.000002S (0706)        STL   F_PB
003515:       02.003612 (0707)       LDA   ='14000
003516:       04.000010S (0708)      STA   F_KEYS
003517:       01.003326 (0709)       JMP   FLTRIN
                       (0710) *
```

Rtn ptr of current stack (Ring 2)

Rebuild last concealed-stack Frame ~~as~~ same as before CALF to point to fault handler

```
03520: 045403.000074X (0711) PTRF3  LDA   XB%+PCSK,1        CSK FIRST
03522:     01.003472  (0712)        JMP   PTRF2
           000074     (0713) PCSK   EQU   '74               CONCEALED STACK PTRS IN PCB
           000010     (0714) LIBMAX EQU   8                 LIBTBL # OF ENTRIES
                      (0715) *
03523: 067430.003371  (0716) PTRNF  EALB  PTRECB+6,*
03525: 061432.000450L (0717)        CALL  ERRPR$
03527: 000100.003613  (0718)        AP    =0,S
03531: 000100.003614  (0719)        AP    =E$FNTF,S
03533: 004400.000050S (0720)        AP    PTRTMP,*
03535: 001500.000001X (0721)        AP    XB%+1,S
03537: 001500.000000X (0722)        AP    XB%,S
03541: 000100.003615  (0723)        AP    =C'PTRFLT',S
03543: 000300.003611  (0724)        AP    =6,SL
                      (0725)
03545: 031410.003571  (0726) BADPTR JSXB  ACCPTR            OUT OF REAL ECBS -- GIVE UP
03547:        150317  (0727)        BCI   8,POINTER FAULT
03550:        144716
03551:        152305
03552:        151240
03553:        143301
03554:        152714
03555:        152240
03556:        120240
              003557  (0728)        ENT   BADGAT
03557: 031410.003571  (0729) BADGAT JSXB  ACCPTR            ENTRY FROM UNUSED ECB IN GATE SEGMENT
03561:        152716  (0730)        BCI   8,UNDEFINED GATE
03562:        142305
03563:        143311
03564:        147305
03565:        142240
03566:        143701
03567:        152305
03570:        120240
03571: 005401.000003S (0731) ACCPTR LDA   F_PB+1            GET PBL
03573: 051422.000452L (0732)        STA   ERRVEC+1          SET ALTVAL(2) = PBL
03575: 061432.000454L (0733)        CALL  ERRRTN
03577: 000500.000002S (0734)        AP    F_PB,S            ALTVAL(1) = PBH
```

```
003601: 000100.003613  (0735)        AP     =0,S                NO ALTRTN
003603: 001500.000000X (0736)        AP     XB%,S               MSG
003605: 000300.003620  (0737)        AP     =16,SL              MSGLEN
                       (0738) *
003607:        00.177774A (0739)     FIN
003610:        00.070000A
003611:        00.000006A
003612:        00.014000A
003613:        00.000000A
003614:        00.000017A
003615:        00.150324A
003616:        00.151306A
003617:        00.146324A
003620:        00.000020A

                       (0740)        EJCT
```

```
            003621    (0741)        ENT    LIBNXT
                      (0742) LIBNXT ECB    LIBN,F_ARG1,2  FS= FLTFS+4

003621:     003632
            000020
            000012
            000002
            177400
            014000

003623:     000054
003632:     000605    (0743) LIBN   ARGT                      (PACKGN,STACK-FRAME)
003633: 045421.000012S (0744)       LDA    F_ARG1,*           PACKGN
003635: 140610.003727  (0745)       BLE    LIBNF              BAD PARAMETER
003637:     11.003737  (0746)       CAS    =LIBMAX
003640:     01.003727  (0747)       JMP    LIBNF
003641:     01.003727  (0748)       JMP    LJBNF
003642:       041477   (0749)       LGL    1
003643:     04.000052S (0750)       STA    PTRX
003644:       140504   (0751)       TAX
003645: 145420.003722  (0752)       LDA    1LIBTBL,*X
003647: 140612.003727  (0753)       BEQ    LIBNF              NO MORE HANDLERS
                       (0754) *
003651: 065432.000446L (0755)       EAXB   PCBSEG             XB -> PCB SEGMENT
003653: 013404.000025P (0756)       LDLR   OWNER
003655:     35.000002A (0757)       LDX    2                  X -> MY PCB
003656: 045403.000075X (0758)       LDA    XB%+PCSK+1,X       CONCEALED-STACK NEXT
003660:       140505   (0759)       TAY                       Y -> CURRENT 6WD ENTRY
003661: 063403.000076X (0760)       CAS    XB%+PCSK+2,X       CONCEALED-STACK LAST
003663:     01.003665  (0761)       JMP    *+2
003664:     01.003724  (0762)       JMP    LIBN3              EQUALS LAST, MUST RESET
003665:     06.003611  (0763)       ADD    =6
003666: 051403.000075X (0764) LIBN2 STA    XB%+PCSK+1,X       SET NEXT PTR
                       (0765) *
003670: 067431.000015S (0766)       EALB   F_ARG2,*
003672: 043426.000002L (0767)       EAL    F_PP-SB%+LB%,*     MOVE INFO TO CONCEALED-STACK
003674: 011437.000000X (0768)       STL    XB%,Y
003676: 005402.000012L (0769)       LDA    F_FCODE-SB%+LB%
```

Reset
concealed
stack

```
003700:        140314   (0770)          TAB
003701: 005402.000010L (0771)          LDA   F_KEYS-SB%+LB%
003703: 011437.000002X (0772)          STL   XB%+2,Y
003705: 005416.000013L (0773)          LDL   F_FADDR-SB%+LB%
003707: 011437.000004X (0774)          STL   XB%+4,Y
                        (0775) *
003711:      35.000052S (0776)          LDX   PTRX           PACKGN
003712: 145434.003722   (0777)          LDL   ILIBTBL,*X     R3 FAULT-HANDLER
003714: 011416.000002L (0778)          STL   F_PB-SB%+LB%
003716:      02.003612  (0779)          LDA   ='14000
003717: 011402.000010L (0780)          STA   F_KEYS-SB%+LB%
003721:        000611   (0781)          PRTN
003722: 000000.000000E (0782) ILIBTBL  IP    LIBTBL
                        (0783) *
003724: 045403.000074X (0784) LIBN3    LDA   XB%+PCSK,1     CSK FIRST
003726:      01.003666  (0785)          JMP   LIBN2
                        (0786) *
003727: 067431.000015S (0787) LIBNF    EALB  F_ARG2,*
003731: 045436.000013L (0788)          LDL   F_FADDR-SB%+LB%,*
003733:        140100   (0789)          SSP
003734: 011415.000050S (0790)          STL   PTRTMP
003736:      01.003523  (0791)          JMP   PTRNF
003737:      00.000010A (0792)          FIN

                        (0793)          EJCT
```

Reset concealed stack

ILIBTL* ~ indexed by package #

INTERRUPTS:

Process Exchange mode on

1) Interrupt from I/O Bus

2) Micro-code

   a) PSWKEYS ← Keys, models

   b) PSWPB ← RP (reg. where instr. ctr. kept when current user)

   c) RP ← Ring Ø, Segment 4, Vector address

   d) Keys ← 64V mode

   e) ICPN - interrupt clear priority network

   f) Set interrup inhibited in keys

   g) Fetch next instruction

3) Next instruction is the beginning Phantom Interrupt code for the interrupt. Phantom interrupt code will either handle the interrupt or cause a process to be scheduled to handle the interrupt.

   Phantom Interrupt code must

   a) Acknowledge the interrupt to the controller

   b) CAI - clear active interrupt

   c) Return from interrupt

EXAMPLE:

MPC Phantom Interrupt Code

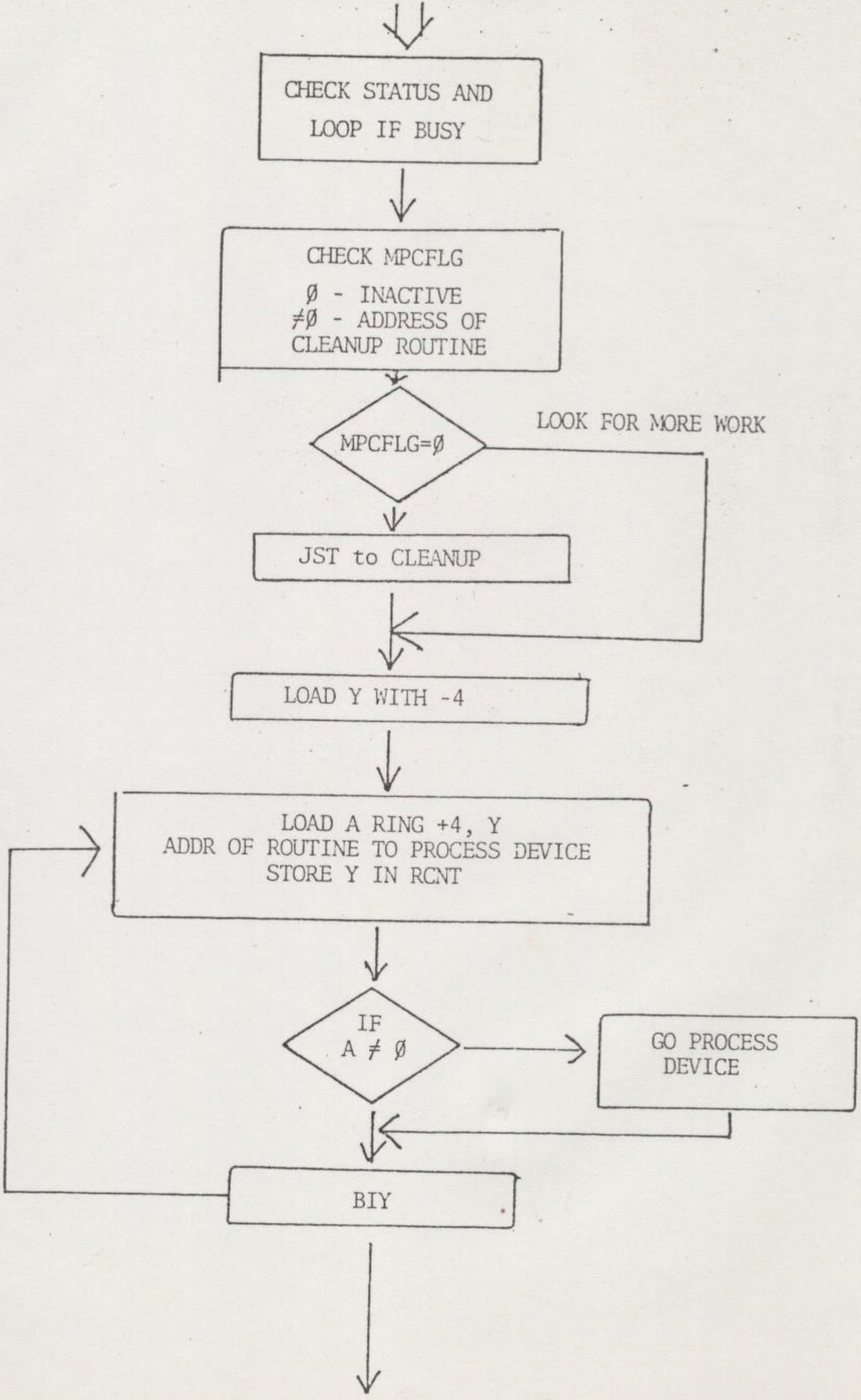
```
                          (0093)    ①
                 000120   (0094)    ↓        ENT     MPCINT    ↙②
000120:   031404.031403P  (0095) MPCINT      OCP     '1403
000122:          001216   (0096)             INEC    MPCSEM         ←③
000123    000000.000506
```

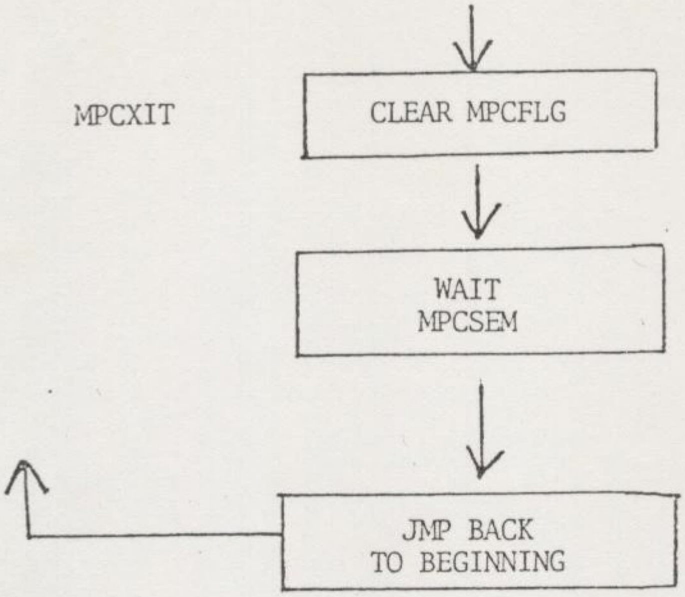
1) Interrup vectors to MPCINT
2) Acknowledge to controller
3) INEC
   - clear active interrupt
   - notify MPCSEM - start interrupt handler proc.
   - return from interrupt

MICRO PROGRAMMED CONTROLLER

MPCDIM

STARTED BY T$xMPC or PHANTOM INTERRUPT CODE, WAITS ON MPCSEM

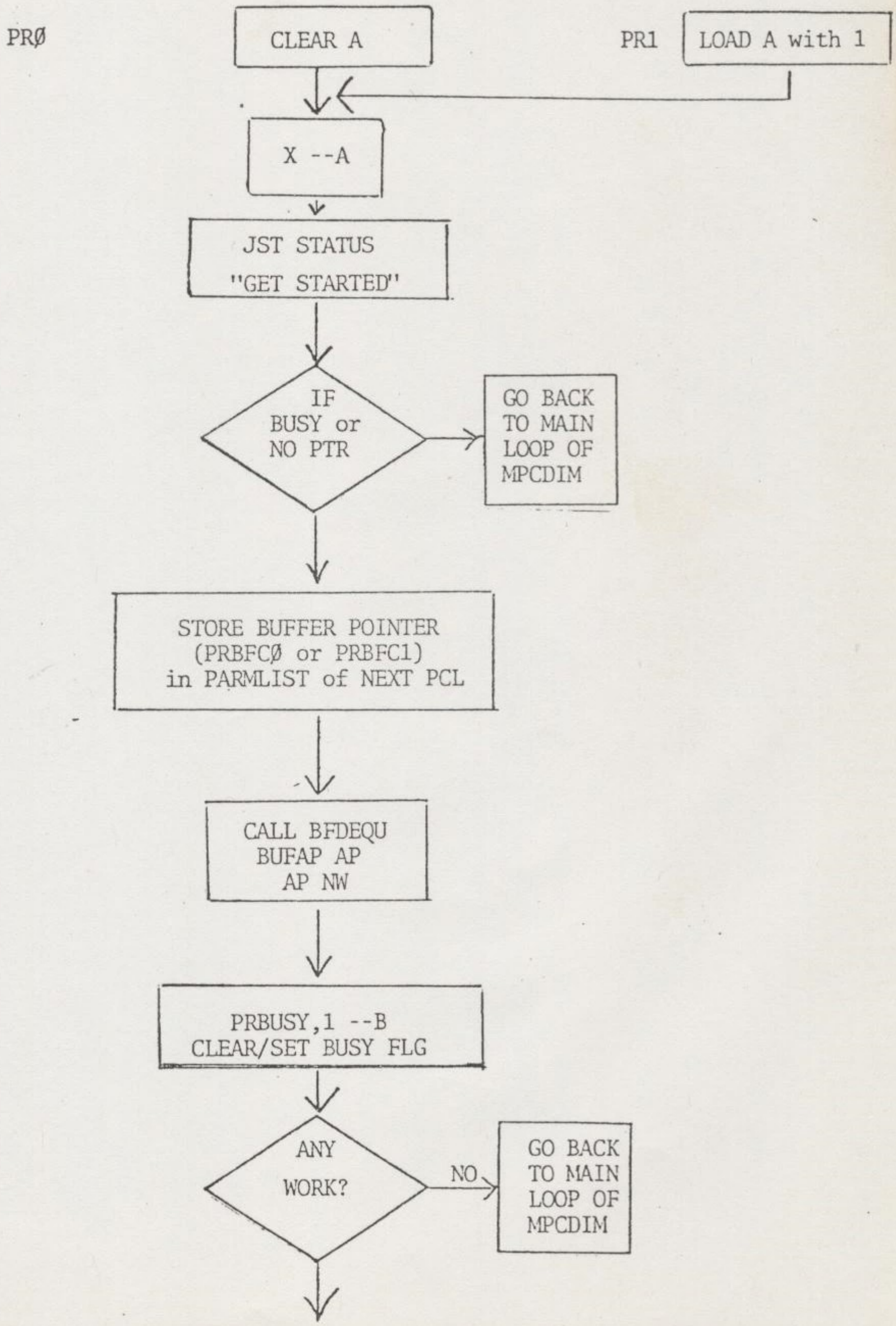


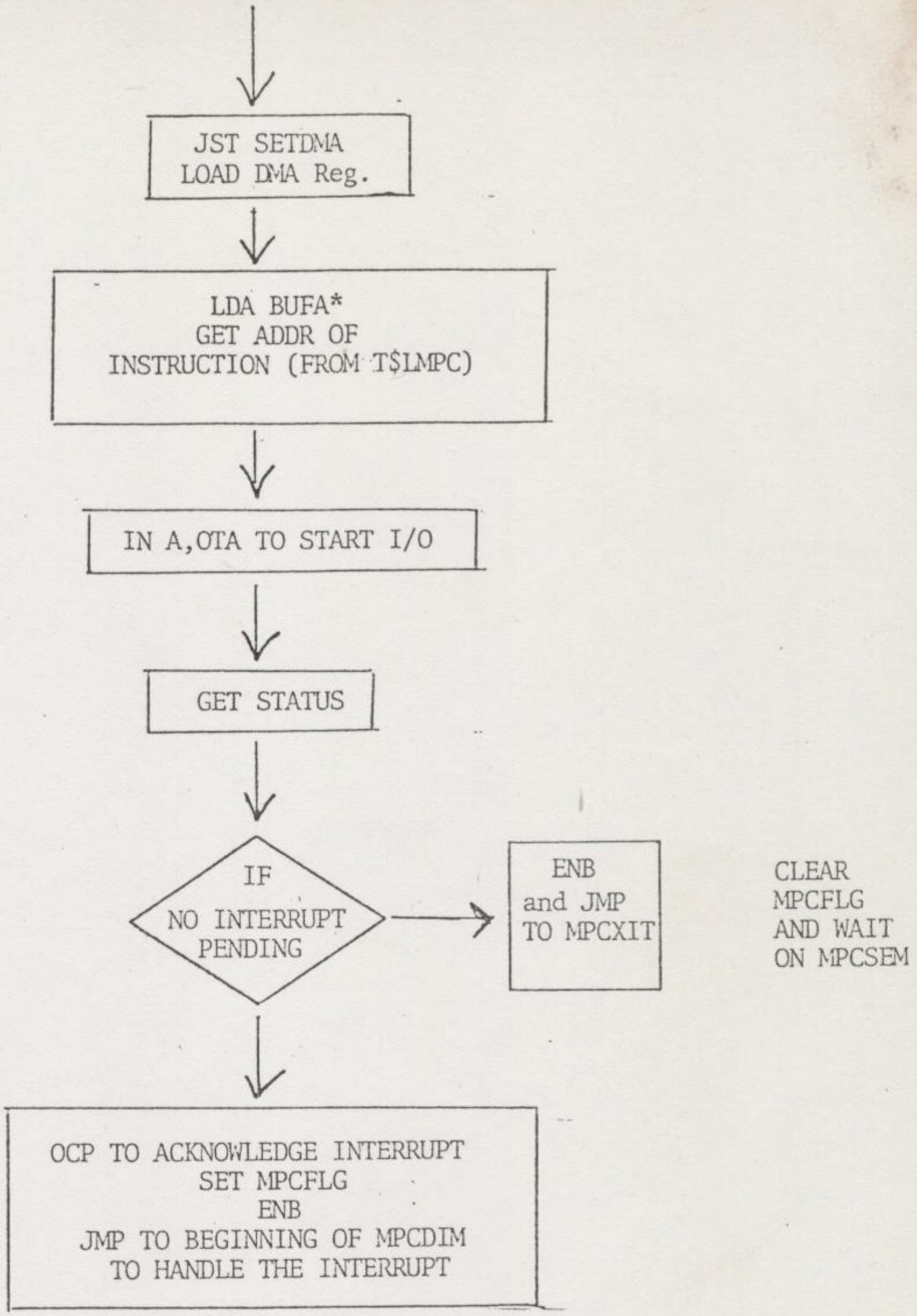

MPCXIT
CLEAR MPCFLG
WAIT
MPCSEM
JMP BACK
TO BEGINNING

PRØ, PR1

PROCESS PRØ, PR1

BRANCHED TO BY MPCDIM

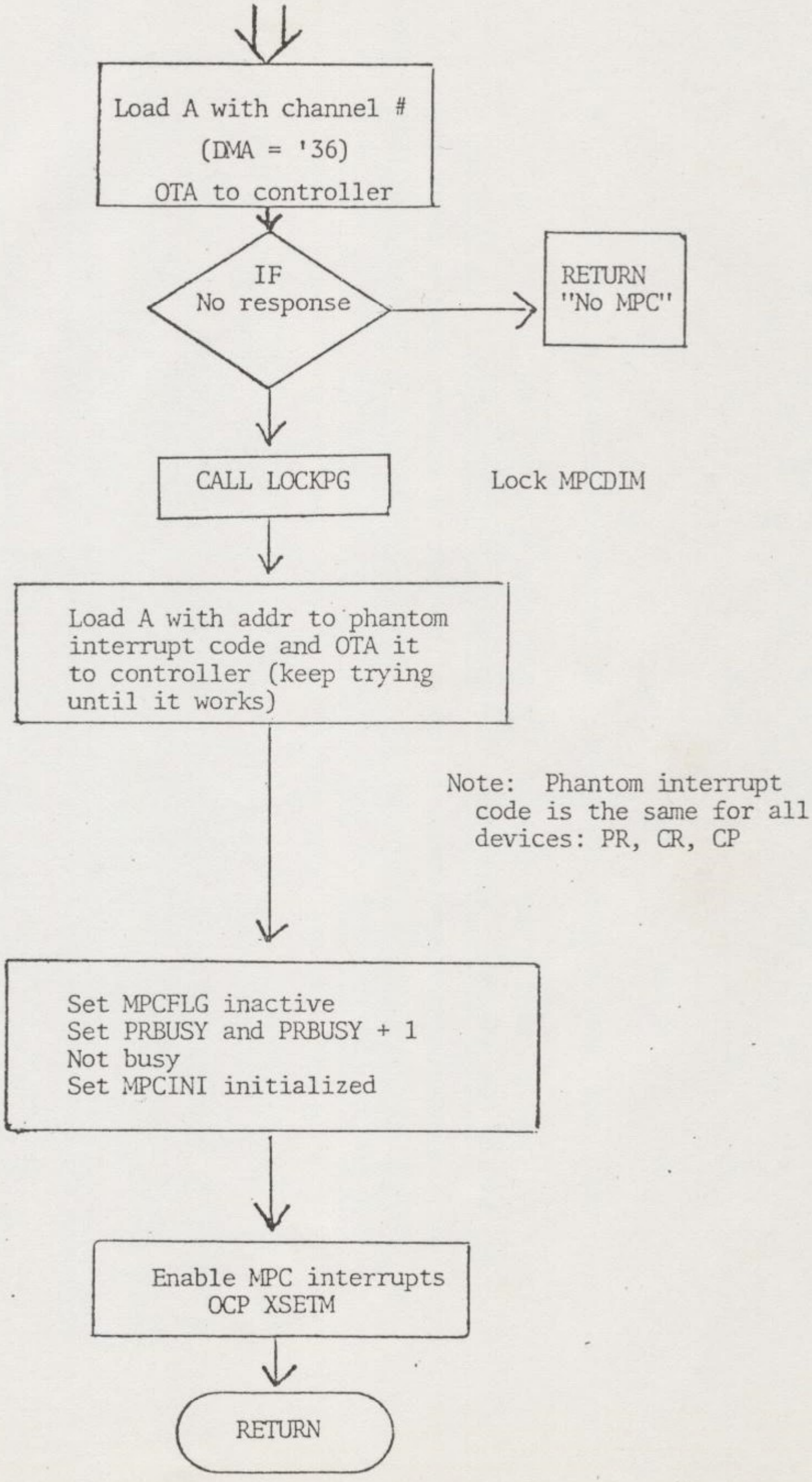



JST SETDMA
LOAD DMA Reg.
LDA BUFA*
GET ADDR OF
INSTRUCTION (FROM T$LMPC)
IN A,OTA TO START I/O
GET STATUS
IF
NO INTERRUPT
PENDING
ENB
and JMP
TO MPCXIT
CLEAR
MPCFLG
AND WAIT
ON MPCSEM
OCP TO ACKNOWLEDGE INTERRUPT
SET MPCFLG
ENB
JMP TO BEGINNING OF MPCDIM
TO HANDLE THE INTERRUPT

MPINIT

MPC INITIALIZATION

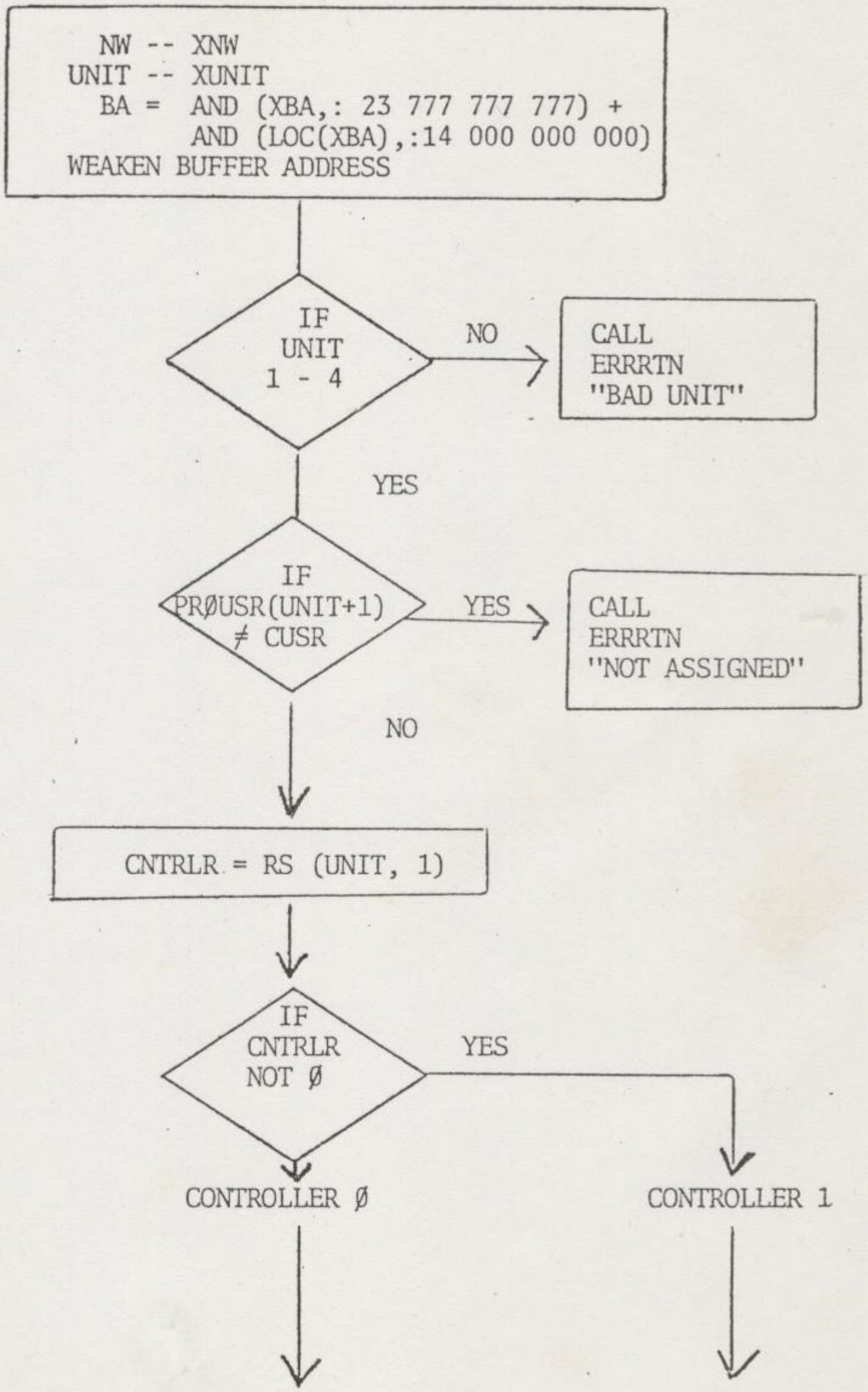
CALLED BY T$xMPC

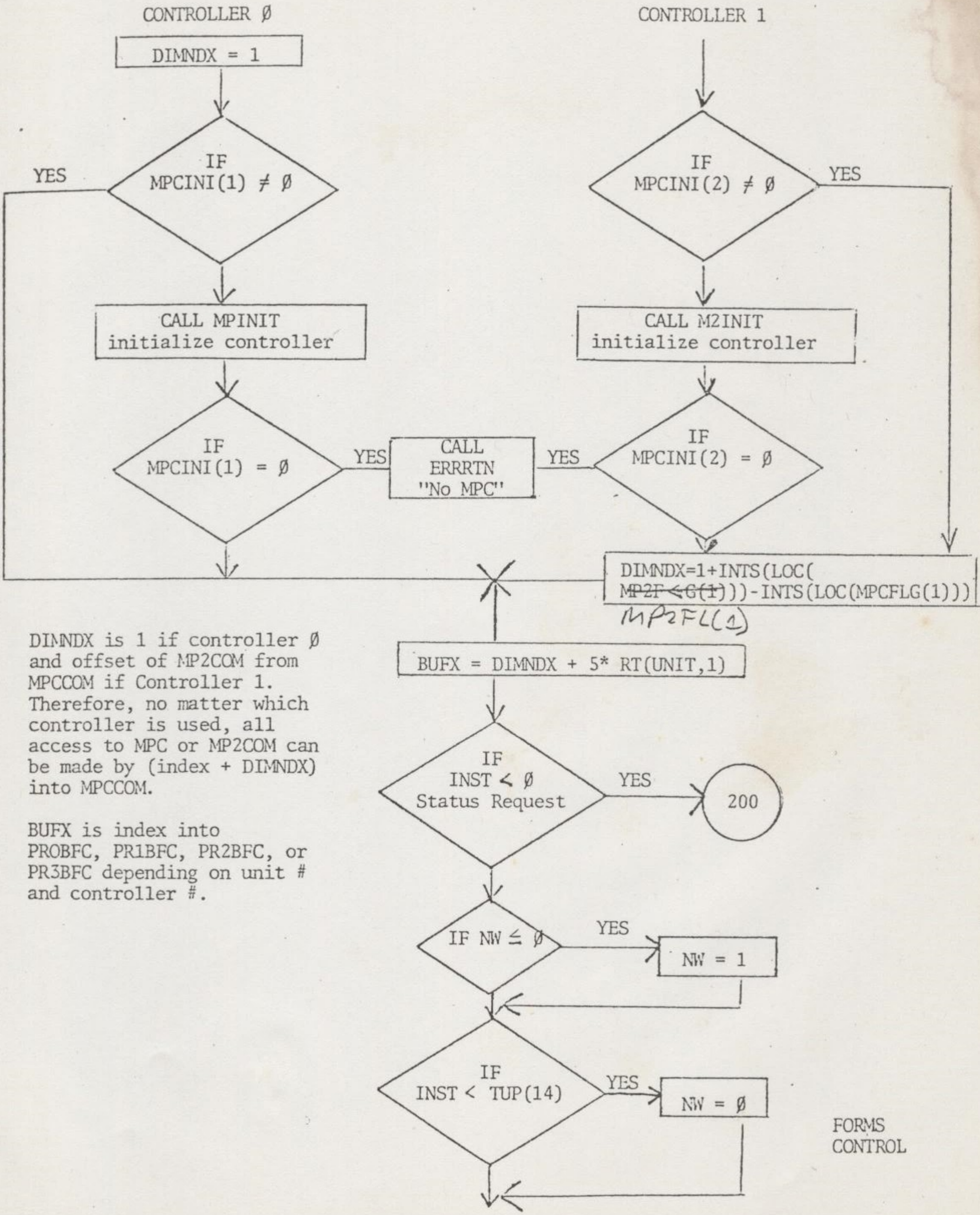


Note: Phantom interrupt code is the same for all devices: PR, CR, CP

T$LMPC - USE ENTRY POINT

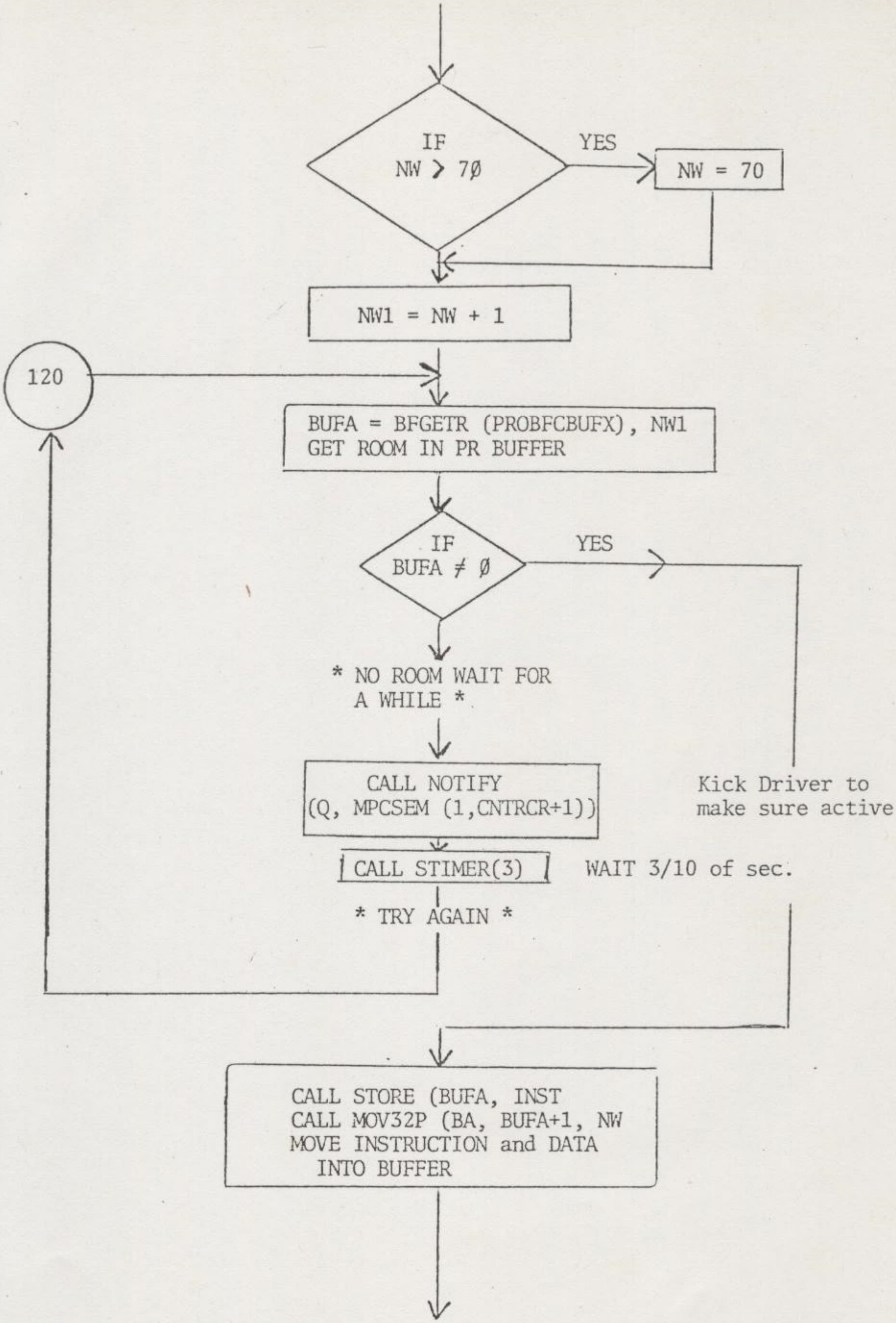
(XUNIT, XBA, ~~XNW,~~ XNw INST, STATV)

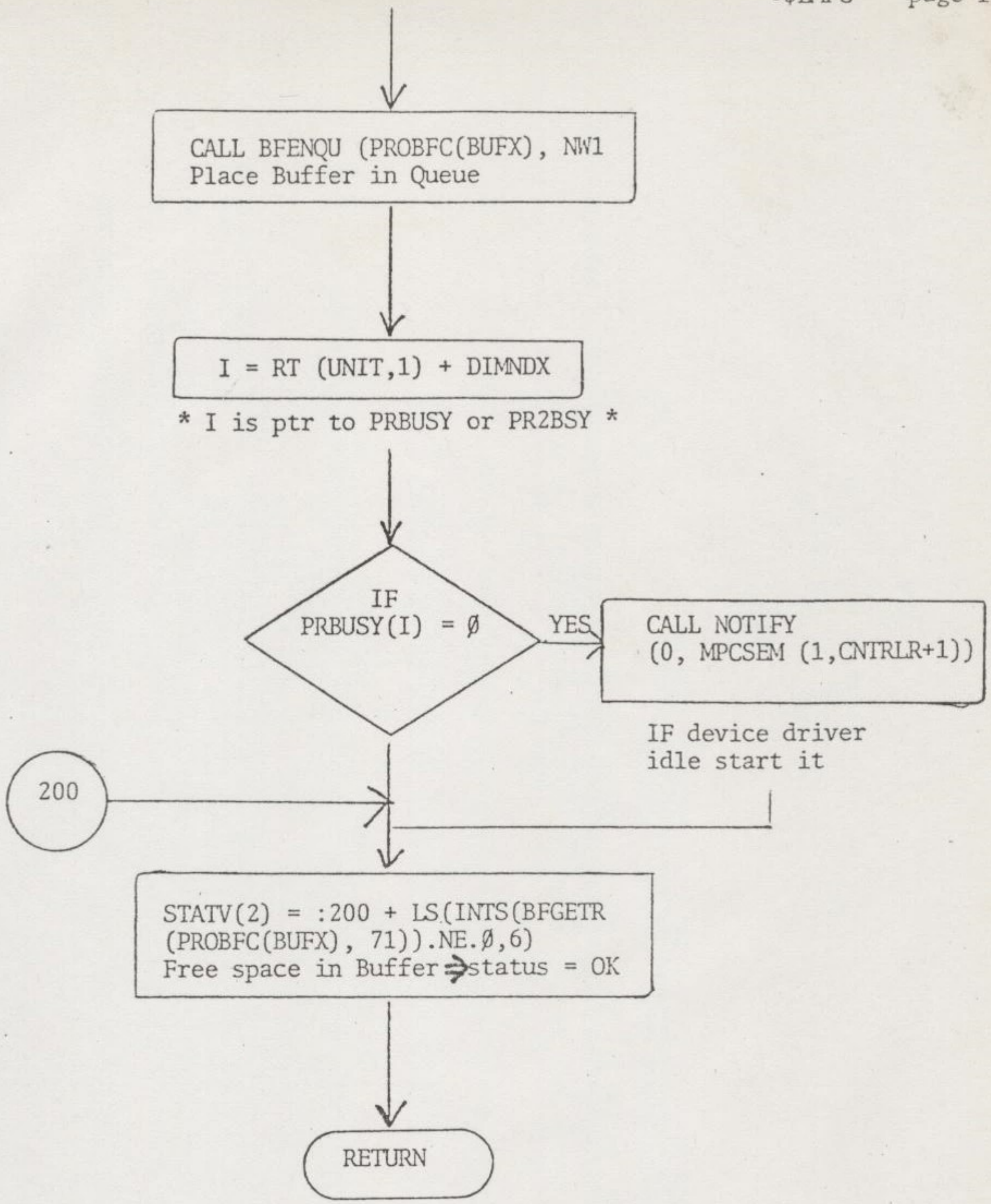




DIMNDX is 1 if controller Ø and offset of MP2COM from MPCCOM if Controller 1. Therefore, no matter which controller is used, all access to MPC or MP2COM can be made by (index + DIMNDX) into MPCCOM.

BUFX is index into PR0BFC, PR1BFC, PR2BFC, or PR3BFC depending on unit # and controller #.


IF
NW > 7Ø
YES
NW = 70
NW1 = NW + 1
120
BUFA = BFGETR (PROBFCBUFX), NW1
GET ROOM IN PR BUFFER
IF
BUFA ≠ Ø
YES
* NO ROOM WAIT FOR
A WHILE *
CALL NOTIFY
(Q, MPCSEM (1,CNTRCR+1))
Kick Driver to
make sure active
CALL STIMER(3)
WAIT 3/10 of sec.
* TRY AGAIN *
CALL STORE (BUFA, INST
CALL MOV32P (BA, BUFA+1, NW
MOVE INSTRUCTION and DATA
INTO BUFFER


CALL BFENQU (PROBFC(BUFX), NW1
Place Buffer in Queue
I = RT (UNIT,1) + DIMNDX
* I is ptr to PRBUSY or PR2BSY *
IF
PRBUSY(I) = Ø
YES
CALL NOTIFY
(0, MPCSEM (1,CNTRLR+1))
IF device driver
idle start it
200
STATV(2) = :200 + LS(INTS(BFGETR
(PROBFC(BUFX), 71)).NE.Ø,6)
Free space in Buffer ⇒ status = OK
RETURN

BFGETR

Get space in Q

BUFA = BFGETR (BUFCON, NW)

BUFA = BUFFER ADDRESS RETURNED

BUFCON = POINTERS INTO BUFFER POOL

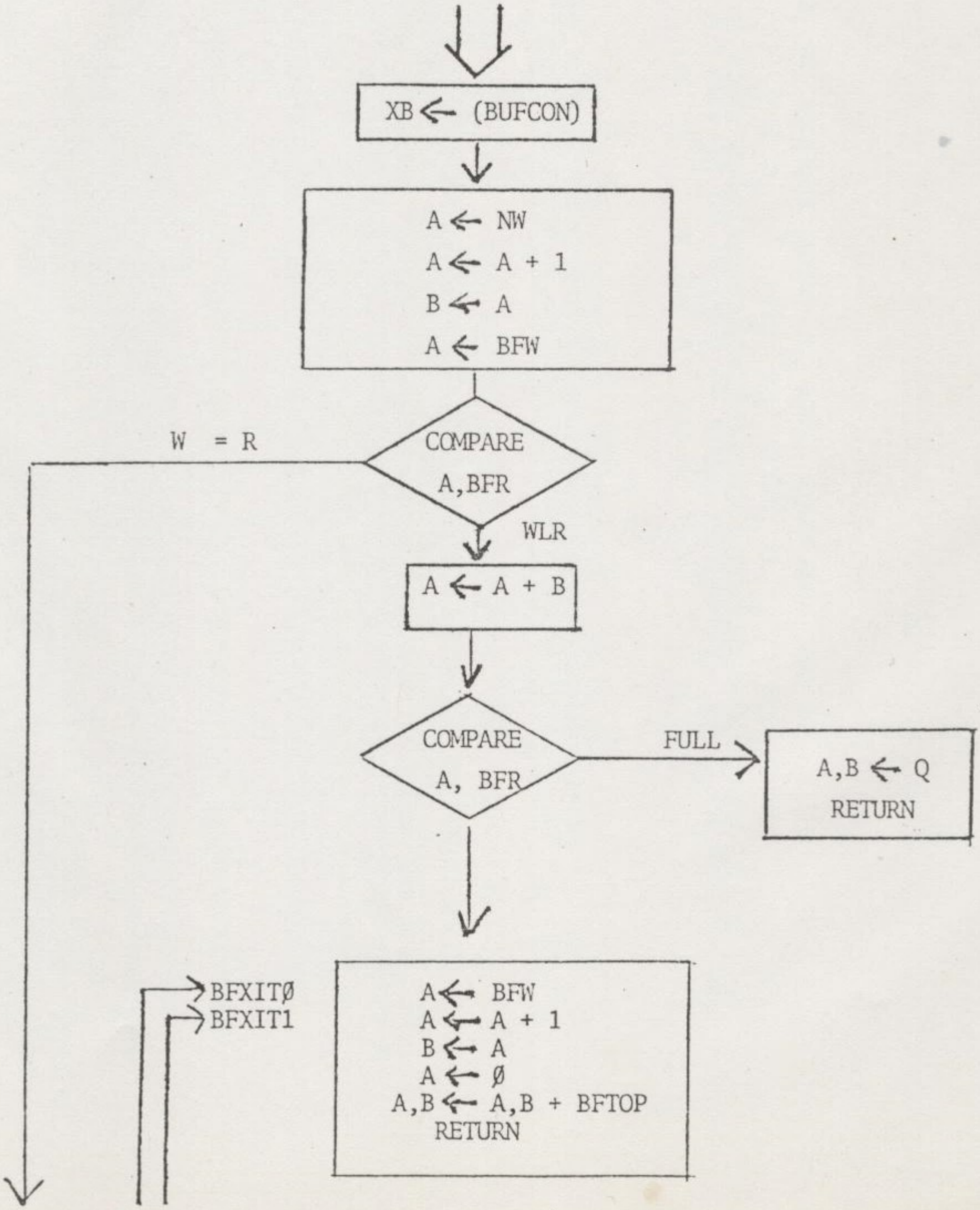
NW = SIZE OF BUFFER WANTED

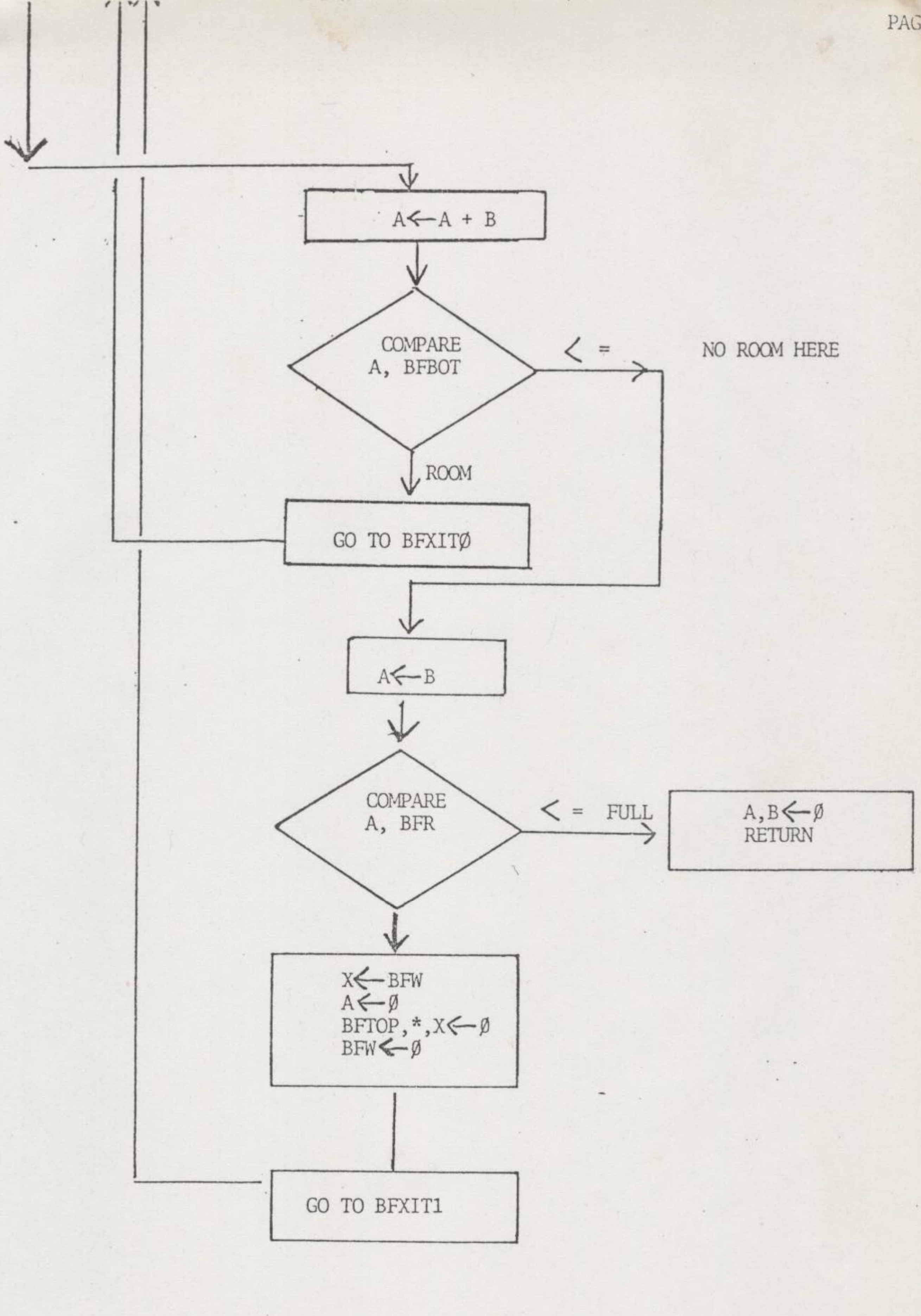
BUFCON + Ø - BFR - read ptr

BUFCON + 1 - BFW - write ptr

BUFCON + 2 - BFTOP - top of Q

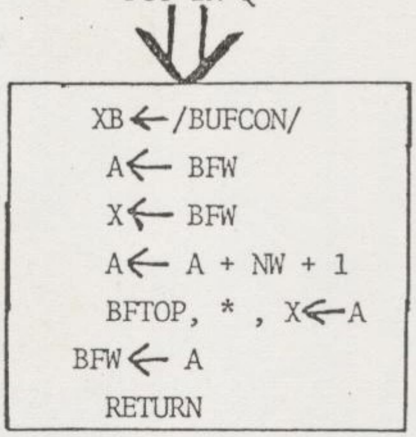
BUFCON + 4 - BFBOT - bottom of Q




A←A + B
COMPARE
A, BFBOT
< =
NO ROOM HERE
ROOM
GO TO BFXITØ
A←B
COMPARE
A, BFR
< = FULL
A,B←Ø
RETURN
X←BFW
A←Ø
BFTOP,*,X←Ø
BFW←Ø
GO TO BFXIT1

BFENQU

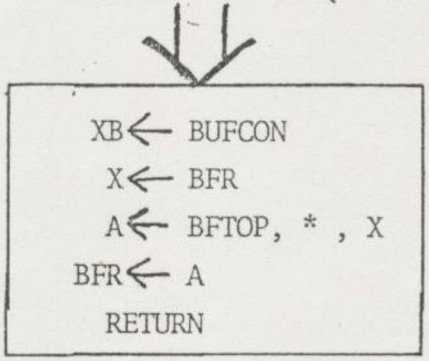
PUT IN Q



```
  XB ← /BUFCON/
   A ← BFW
   X ← BFW
   A ← A + NW + 1
   BFTOP, * , X ← A
BFW ← A
   RETURN
```

BFRELS

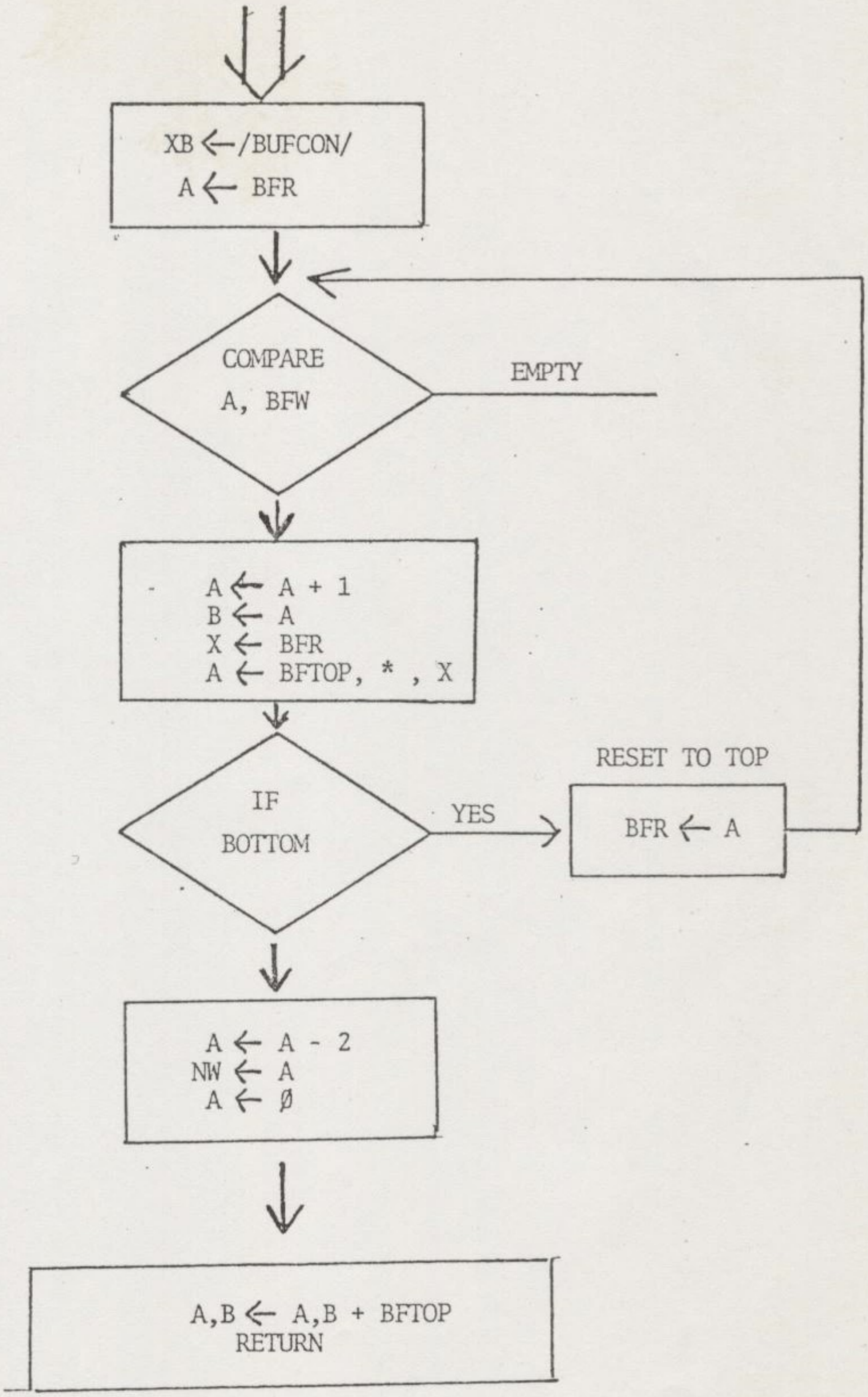
Release ITEM in Q

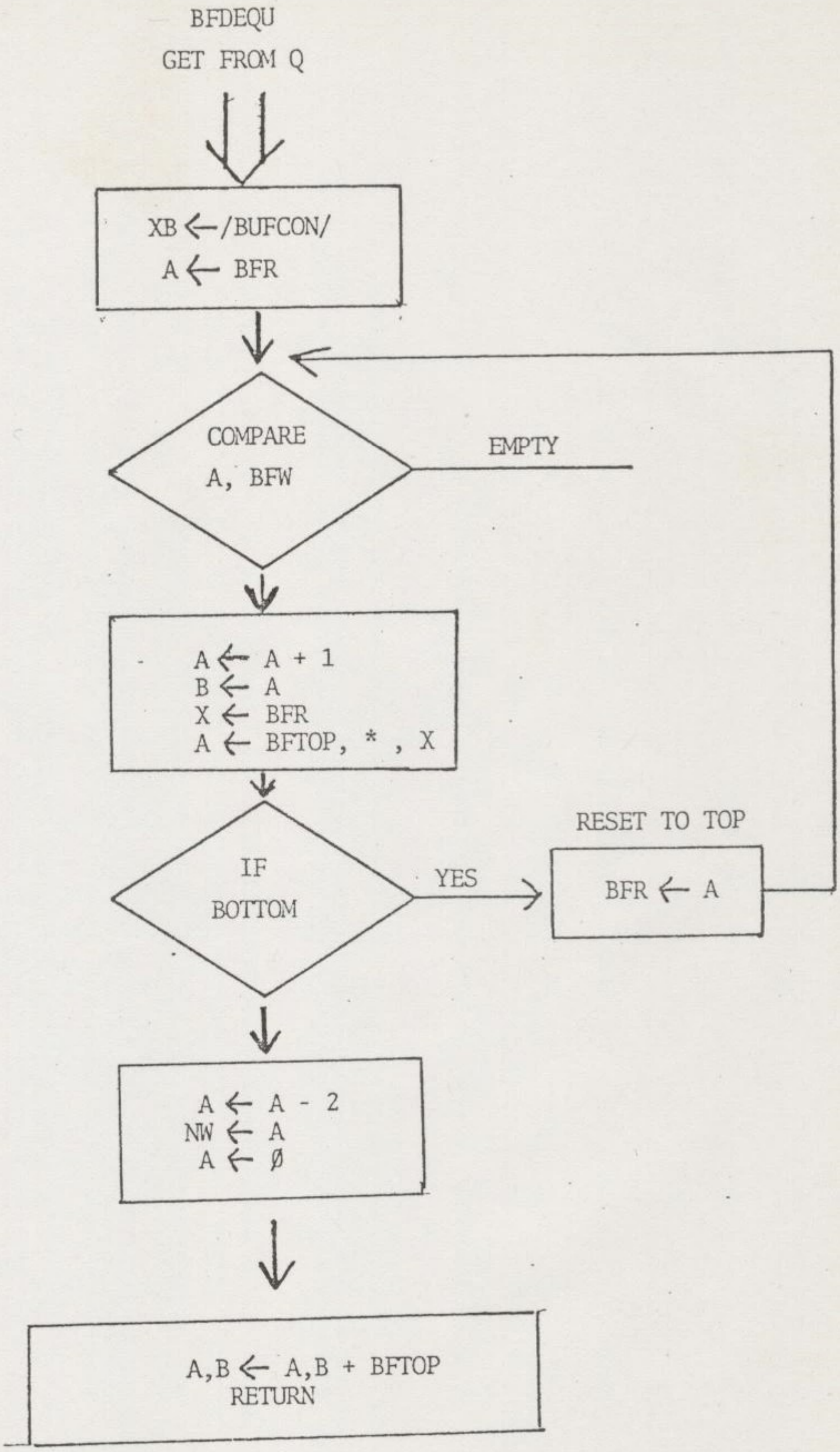


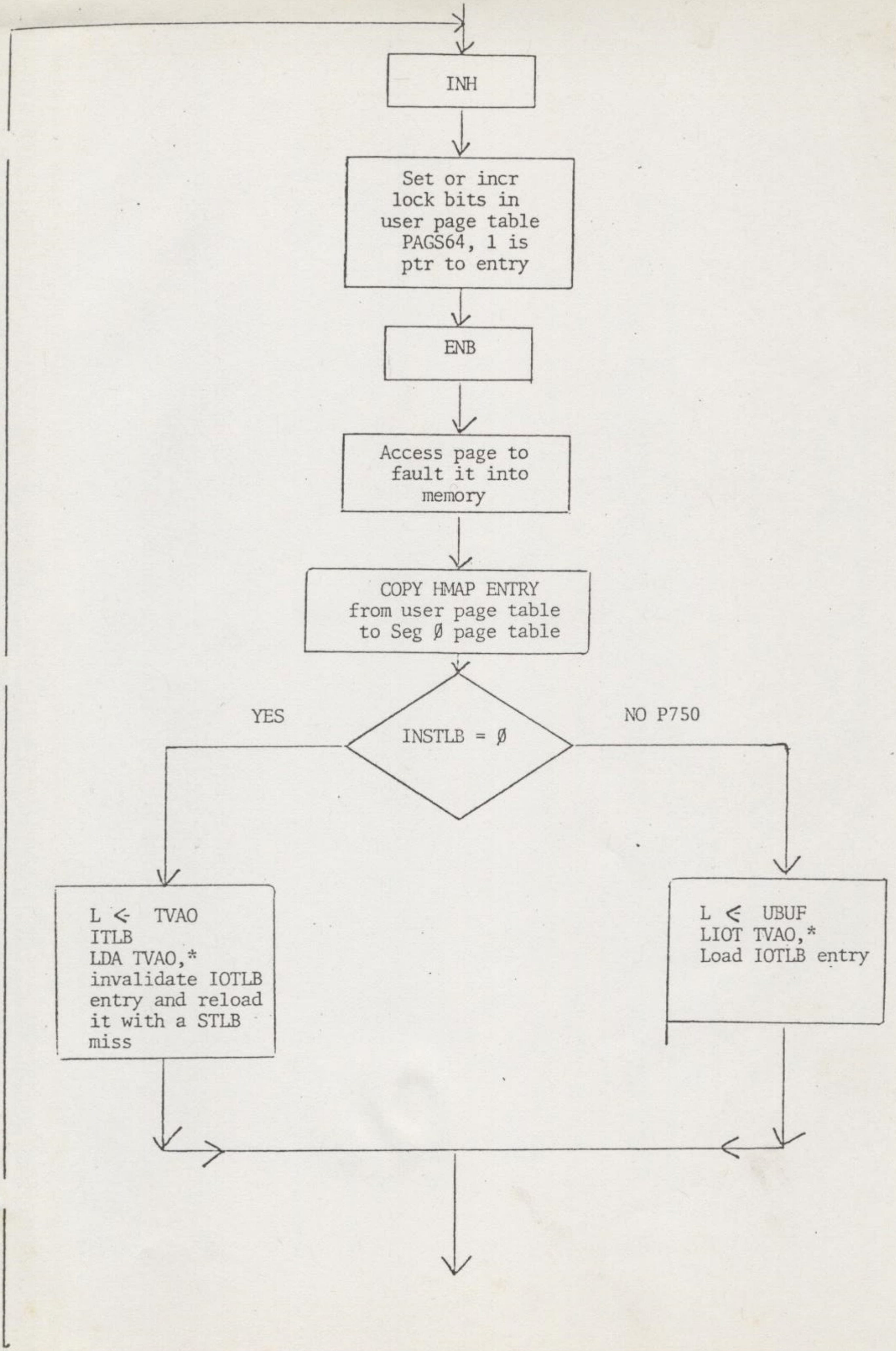
```
  XB ← BUFCON
   X ← BFR
   A ← BFTOP, * , X
BFR ← A
   RETURN
```

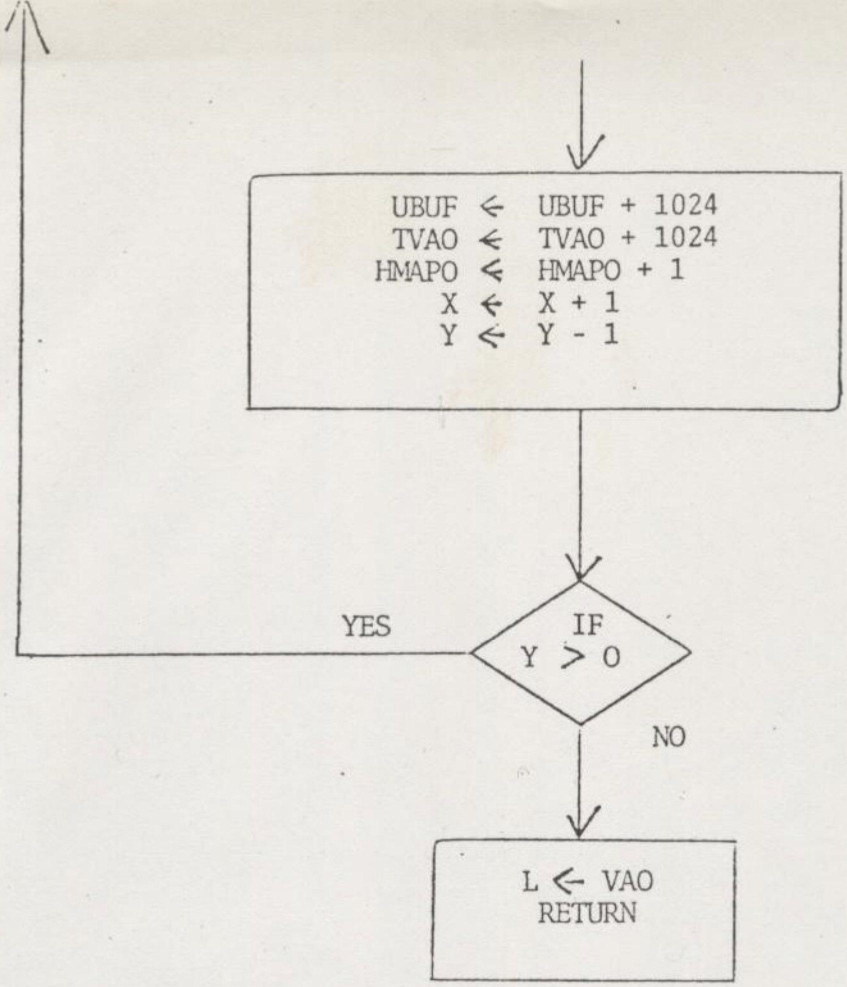
BFDEQU

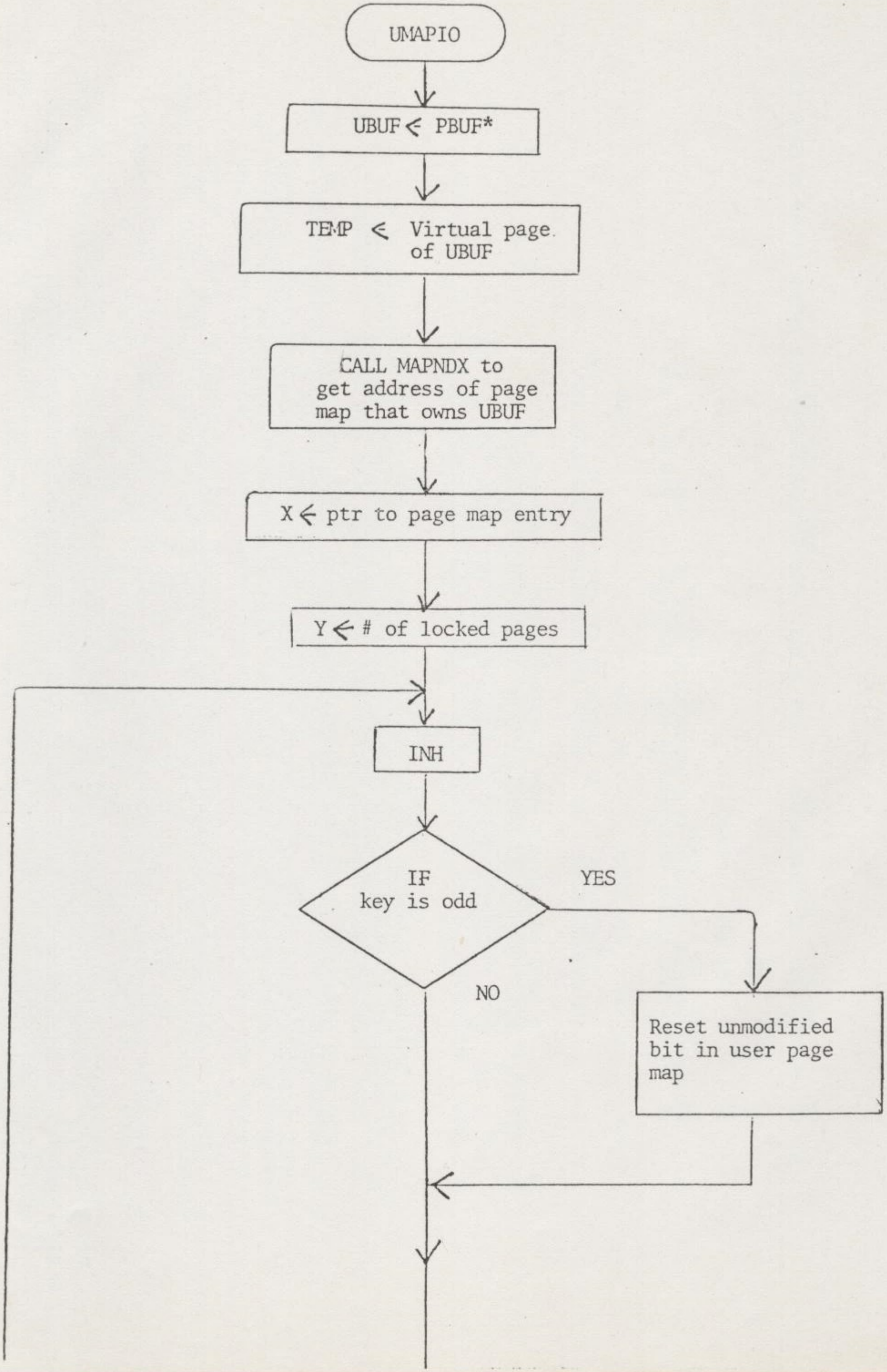
GET FROM Q

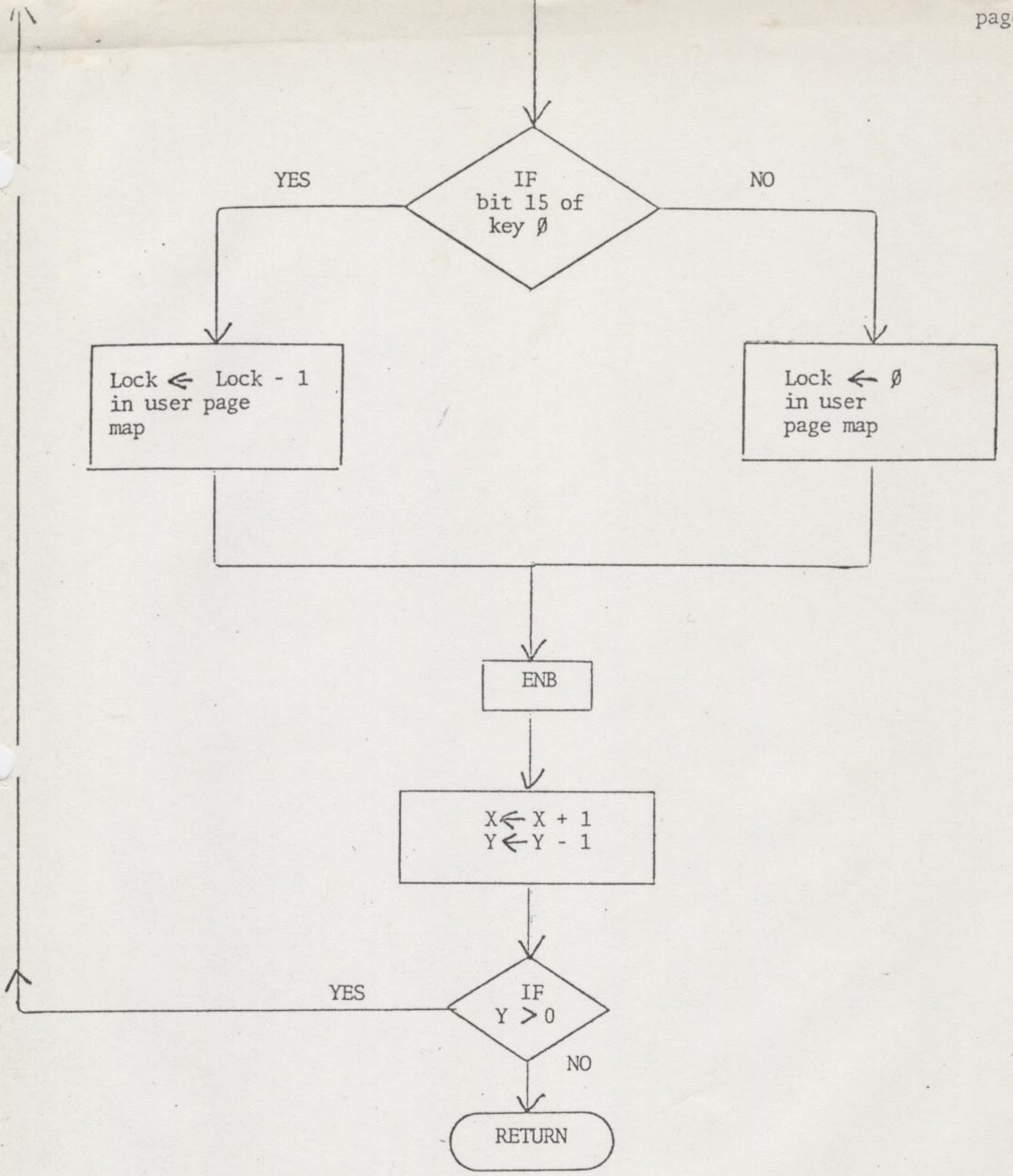



BFDEQU
GET FROM Q
XB ← /BUFCON/
A ← BFR
COMPARE
A, BFW
EMPTY
A ← A + 1
B ← A
X ← BFR
A ← BFTOP, * , X
IF
BOTTOM
YES
RESET TO TOP
BFR ← A
A ← A - 2
NW ← A
A ← Ø
A,B ← A,B + BFTOP
RETURN


INH
Set or incr lock bits in user page table PAGS64, 1 is ptr to entry
ENB
Access page to fault it into memory
COPY HMAP ENTRY from user page table to Seg Ø page table
YES
INSTLB = Ø
NO P750
L <- TVAO
ITLB
LDA TVAO,*
invalidate IOTLB entry and reload it with a STLB miss
L < UBUF
LIOT TVAO,*
Load IOTLB entry


UBUF ← UBUF + 1024
TVAO ← TVAO + 1024
HMAPO ← HMAPO + 1
X ← X + 1
Y ← Y - 1
YES
IF
Y > 0
NO
L ← VAO
RETURN


UMAPIO
UBUF ⇐ PBUF*
TEMP ⇐ Virtual page of UBUF
CALL MAPNDX to get address of page map that owns UBUF
X ⇐ ptr to page map entry
Y ⇐ # of locked pages
INH
IF key is odd
YES
NO
Reset unmodified bit in user page map


IF
bit 15 of
key Ø
YES
NO
Lock ← Lock - 1
in user page
map
Lock ← Ø
in user
page map
ENB
X ← X + 1
Y ← Y - 1
IF
Y > 0
YES
NO
RETURN

PRIME COMPUTER INTERNATIONAL

REFERENCE NOTES ON THE AMLC

PREPARED BY: C PARTRIDGE

NOVEMBER 1978

CONTENTS

1) INTRODUCTION

This document is designed as an aid to using and understanding the AMLC hardware and software.

The standard documents describe the use of the AMLC related commands, but a description of how the software and hardware works can only be found in internals course notes, which really require attendance on the course.

Many problems occur in normal usage of the AMLC due to a lack of knowledge of how best to use the system. When it comes to making a modification to the software to adapt it for a special requirement, all nature of problems occur.

The information contained in this document is split into a number of sections:

a) A brief description of the AMLC.

b) The user commands and what they do.

c) A more detailed view of the software.

d) Interfacing special devices and coping with known bugs.

e) Differences on the P300.

The information refers to the segmented architecture. The differences in the P300 are described in Section 6.

The details refer to the Rev 15 and Rev 16 releases of PRIMOS.

2) BRIEF DESCRIPTION OF THE AMLC

2.1 The Hardware:

The AMLC (Asynchronous Multiline Controller) interfaces full duplex/half duplex data lines to a PRIME computer.
There are basically three types of boards:
5002, 5004 half duplex
5052, 5054 full duplex
5152, 5154 full duplex with QAMLC
The last digit refers to the number of lines (2 = 8, 4 = 16).
The half duplex type isn't supported by standard software.

A P300 can handle 2 boards (not QAMLC type). A P350, 400, 500 can handle QAMLC with a 400, 500 expandable to 4 boards.

Information is transferred by Programmed Input Output (PIO), interrupt and DMX transfer. PIO is used for setting states or reading control words. Information transfer is achieved on the standard board by DMC on input and DMT for output. The QAMLC board uses DMC for input and DMQ for output. The speed of a line may be altered by software as can the character format and parity.

2.2 The Software

The components of the software for the AMLC are:

a)	The AMLC driver AMLDIM	(Segment 6)
b)	The AMLC phantom interrupt code	(Segment 4)
c)	The user ring buffers	(Segment 7)
d)	The input tumble tables	(Segment Ø)
e)	The dedicated calls	(Segment Ø)

The software uses two basic mechanisms. The first one, DMX transfer occurs without direct software intervention. The second one, interrupt processing involves a) and b).

The design aim is to reduce the overheads incurred with the 2nd mechanism because this software is of course consuming CP power.

2.3 DMX Transfer

This mechanism uses cycle stealing. This means that the flow of execution is not affected while DMX is going on. However, in the micromachine which is where the microcode comprising each instruction is being executed, thare is a temporary break to handle the DMX service. This microcode is known as firm wear.

Incoming characters from the device use Direct Memory Control. This method uses a pair of pointers in memory to indicate a memory area where characters can be placed. Each AMLC board has two such pointer pairs and memory areas (known as tumble tables). At Cold Start, the AMLC board (the controller) is loaded with these pointer pairs, and triggered. For a system with 4 boards there are consequently 8 tumble tables. Each tumble table is 48 words long. Characters arriving from a device are routed to the tumble table. The 2 byte (1 word) entry consists of a line number and the character, or a bit pattern in the line number byte to indicate a condition ie: break. This process continues until the tumble table is full. At this point, the controller signals this fact (interrupts) and switches input to the other tumble table. This toggling action continues automatically. 'It is the responsibility of the software to remove these characters before the toggle action overwrites the table.

Outgoing characters can use one of two mechanisms:

a) DMT (Direct Memory Transfer)

b) DMQ (Direct Memory Queue)

DMT is the most common mechanism. In memory, a cell is maintained for each line. The controller is given the address of the cell block. Each cell is scanned at the rate for the line pertaining to the cell, for presence of a character. If a character is present, it is moved to the output device and the cell cleared by the controller. It is the responsibility of the software to fill the cells at a sufficient rate to satisfy the line speed to which the cell relates.

The second mechanism, DMQ is available on the 51 series boards. With this technique, the dedicated cell is replaced by a queue. It is the responsibility of the software to top up the queue before the AMLC has extracted all the characters at the line speed.

2.4 Interrupt Processing

Transfers to and from memory occur without software interruption. It is the responsibility of the software to remove the characters from the tumble tables at a fast enough rate and place characters in the dedicated cells or queues to satisfy the line speeds. The software is invoked by means of interruption from the controller. Each line on the controller has a flag bit called the Character Time Interrupt flag (CTI). If this flag is enabled then an periodic interrupt is generated by the AMLC at the rate for the line. The worst situation could be every line going at 9600 baud with the CTI flag on. In this case it is unlikely that the CPU would do anything apart from running AMLDIM, trying to service this interrupt rate. This state of affairs is avoided in a balanced system by using the CTI flag in an ordered manner. For input the CTI flag is set on a particular line at a low rate. This nominated line, called the input clock line, (one for the whole system) is set to interrupt 10 times per second

At this rate, software examines the tumble tables and removes the characters. This is fine while the input rate is low (human type speed). A second mechanism exists to handle the case where characters are coming in more rapidly ie: a fast device sending in characters. When a tumble table is full, the AMLC recognises this and generates an interrupt known as an End of Range (EOR) interrupt. This causes the software to clear the tumble table, hopefully before the other tumble table fills, (which, of course, happens normally). These two mechanisms cope with the two extremes. The first one, typing a few characters at one terminal, ensures that the characters are interpreted by PRIMOS and not just left in the tumble table until an EOR is eventually generated. The second one, flooding the AMLC with characters, prevents data loss except in the limiting case where the input rate is greater than the ability of the software to handle it.

For output the CTI flag is set on a particular line at a faster rate than input. This line is called the output clock line, (one for the whole system). For the DMQ case: A single clock line controls output and input. In the DMT case the software examines the dedicated cells of all the lines and fills up any that are zero if characters are available. In the DMQ case, the software tops up the queues if possible. This system is fine if the lines are operating at the output clock line speed (or lower) in the case of DMT. If it is desired to run the line at a high speed, then two techniques are available. The first one is to make the output clock line run at the high speed. The disadvantage of this is that the amount of CP power required to service this rate increases. At 9600 baud the CPU can spend a large percentage of time (>50%) checking the dedicated cells, if this technique is adopted. The second technique is to switch on the CTI flag for the particular line. However when no more characters are to be transmitted, then the flag must be switched off (otherwise the overheads approach the first method).

Normally the second method is adopted. The first one is usually only chosen by accident. With DMQ high speed lines are handled by increasing the size of the queue so that the topping up of the queue 10 times a second can cope with the higher rate. In practise it is difficult to drive a line at the maximum rate of 9600 baud due to machine loading.

2.5 Software Implementation

The previous section described the software mechanisms that are operating system independant. In other words, the interrupt processing is not dependant on the type of operating system. If the system has an AMLC board, then the software must perform the required servicing. This section describes the software conventions adopted by PRIMOS to interface the AMLC to the rest of the system.

The first consideration is the eventual destination of incoming characters and the store where outgoing characters reside. Each configured line (terminal users and assigned lines) has an input and an output buffer. These buffers are circular (ring) and default to 192 characters on input and 384 characters on output. Characters arrive at the input buffer from a device at the rate the device is transmitting. When the buffer is full, echo back is disabled. User space programs remove characters from the buffer using normal input read routines. Characters arrive at the output buffer from user space programs. When the buffer is full, the user is suspended. Associated with each line is a data word called the LWORD. This is used by the software to determine which buffer is being used for the line and various characteristics set for the line.

Note echo is achieved in the software not in the controller.

At cold start time, a test is made to see how many boards are plugged into the system. The internal tables are adjusted according to the result. The last line is called the group 1 line and determines the rate at which the tumble tables are scanned. The next line back is called the last line of group Ø and determines the rate at which the dedicated cells are scanned for output. In a DMQ system, there is no group 1 and the clock line becomes the last physical line.

3) THE USER COMMANDS

This section describes the commands that affect the AMLC and its associated software. The user has to be the supervisor (system console) except for the ASSIGN and TERM command.

3.1 AMLC

This is the major command affecting the AMLC. It is issued from the system console either "on the fly" or in the C ←PRMO file. The format is:

AMLC (protocol) line number (config) (Lword)

The variants are:

i) AMLC protocol line number config

ii) AMLC protocol line number config Lword

iii) AMLC line number config

iv) AMLC line number config Lword

v) AMLC protocol line number

The protocol may be TRAN, TRANHS, TTY, TTYHS, TTYNOP. The HS protocols invoke the CTI bit on output. Consequently these are used if the line is being set to a speed greater than the output clock line. For DMQ systems HS must not be used. The difference between TRAN and TTY concerns the treatment of newline characters, the parity bit and echo.

For TTY protocol carriage return is echoed for line feed, bit 8 is set true and the character is echoed unless specified otherwise in LWORD. TTYNOP disassociates the line from a user space and it is used when:

a) A USRASR space is being set up and can be used to achieve:
b) An assigned line is being set up

In case a) the line being no opped is 2 less than the user number. Case b) is usually specified if transparent protocol is being used. The line number is specified in octal. The config word is a bit pattern used to set up line speeds, stop bits and character length. On receipt of the config word, PRIMOS issues a PIO to the controller to alter its state. The speed bits have 4 fixed speeds, a programmed clock and 3 jumper assignable speeds. The programmed clock is usually set to 9600 baud. The jumpers have to be set on a complete board basis. Normally installations choose the intermediate speeds between 1200 baud and 9600 baud. The LWORD controls treatment of carriage return, echo and XON/XOFF. The right hand byte determines whether the line is associated with a user space. To make a line assignable, this byte must be cleared. The exact specification of the config LWORD bit pattern can be found in the System Administrators Guide.

3.2 . ASSIGN/UNASSIGN

This command is used when it is required to assign an AMLC line. It is issued from user space. It uses the same format as AMLC, the ASSIGN/UNASSIGN being placed before AMLC, ie: AS AMLC etc.

Two important points to note are:

a) LWORD can not be altered from user space.

b) Not specifying the protocol will default the line to TRAN.

The implications of a) are that features like XON, if set up this way, have to be done on the LWORD attached to the original AMLC command input at the system console. The implications of b) are that if a feature like XON is required, then TTY or TTYHS must be specified because XON will not work under TRAN. For the UNASSIGN, an abbreviated syntax is allowed, ie: UN AMLC lineno.

3.3 AMLBUF

This command can only be issued at cold start from the CONFIG data file. It is used to change the buffer sizes and the Queue size if DMQ is being used. Note, however, that the latter doesn't work under Rev 15. The parameters are octal words, so for buffer sizes, a conversion to decimal characters has to be made, eg: a parameter of 1000 would give a buffer of 1024 characters. The line number is also octal.

Problems occur if AMLBUF is being used to alter assigned lines. The line number must be the next one beyond the terminal lines for the 1st assigned line and the one above that for the next and so on. This is because the buffer given to an assigned line is taken from a pool residing above the terminal buffers. The order in which the buffers are given is determined by the order in which the lines are assigned. The physical line is not used for these calculations. Imagine a system where NUSR = 4 and NAMLC = 3. The AMLBUF command must use line number 3 for the 1st assigned line, 4 for the 2nd and 5 for the 3rd. The line actually assigned is immaterial.

When using the DMQ parameter, the queue size must be calculated 2**K, $4 \geq K \leq 16$ If the queue size is less than 16, then a machine halt will occur.

3.4 NUSR

This command controls the number of terminal lines configured for this session. NUSR must be placed in the CONFIG data file. NUSR which is octal, represents the number of users including the system user.

3.5 NAMLC

This command controls the number of available AMLC lines. Buffers are locked according to the combination of NUSR and NAMLC.

3.6 TERM

This command alters the characteristics of the AMLC from user space. It makes the LWORD bits available at user space, in particular XON/XOFF and duplex. TERM will clear bits 4 - 8 of LWORD so, if these bits have been used by a modified system, then care must be exercised.

4) INNER DETAILS OF THE AMLC SOFTWARE

This section is intended to give an indepth view of the software. If it is required to hang devices on the AMLC or modify the software for specials then the implications of doing this have to be understood so that unpredictable side effects are not experienced.

4.1 Overview

The most important module handling the AMLC is AMLDIM. This module runs as a complete process and has its own semaphores to control the character flow. AMLDIM is where control goes eventually when an interrupt is received. This module uses a number of other modules:

i) FMLIOB (From Logical Input Output Buffer). This module is responsible for obtaining characters from the ring buffer and passing them to AMLDIM.

ii) TOLIOB (To Logical Input Output Buffer). This module is responsible for placing characters in the ring buffer (either input or output).

iii) BUFCHK. This module examines the ring buffer to see if there is room for a given number of characters.

The code that handles the interrupt is contained in SEG 4. This code causes the interrupt response code (IRC) to be invoked.

4.2 Phantom Interrupt Code (PIC)

When an interrupt is received by the microcode, control passes to a location in segment 4. The current PB register and KEYS are saved by the microcode and the code located in segment 4 is executed.

For the AMLC this code consists of 5 instructions. There are 4 OCP instructions and an INEC AMLSEM. The OCP instructions clear the AMLC's interrupt mask and disable any further interrupts. The INEC is a process exchange instruction that:

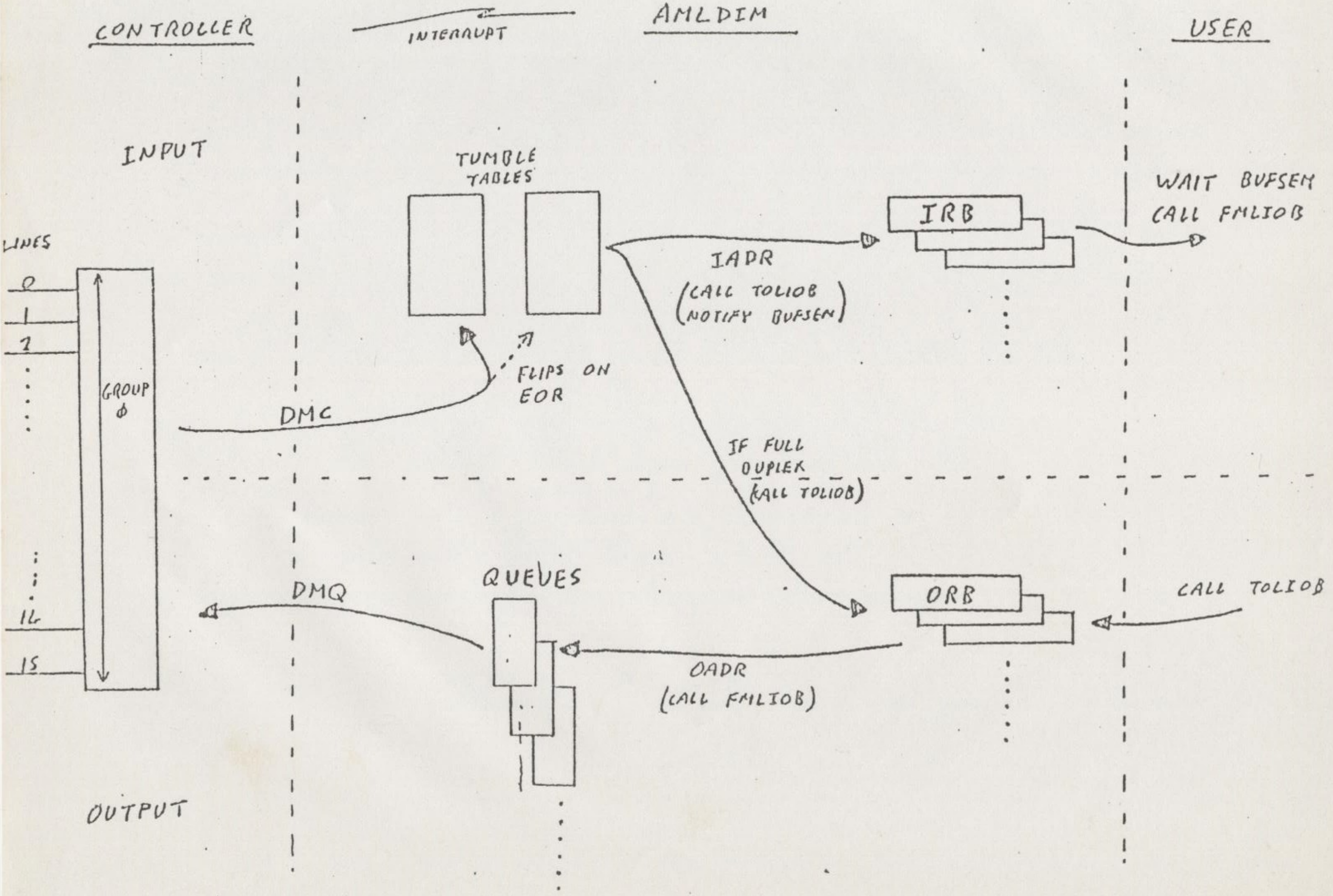
i) Notifies the semaphore AMLSEM and places the PCB on that semaphore on the end of the ready list at correct level.

ii) Issues a CAI operation which frees the backplane of the CPU for further interrupts.

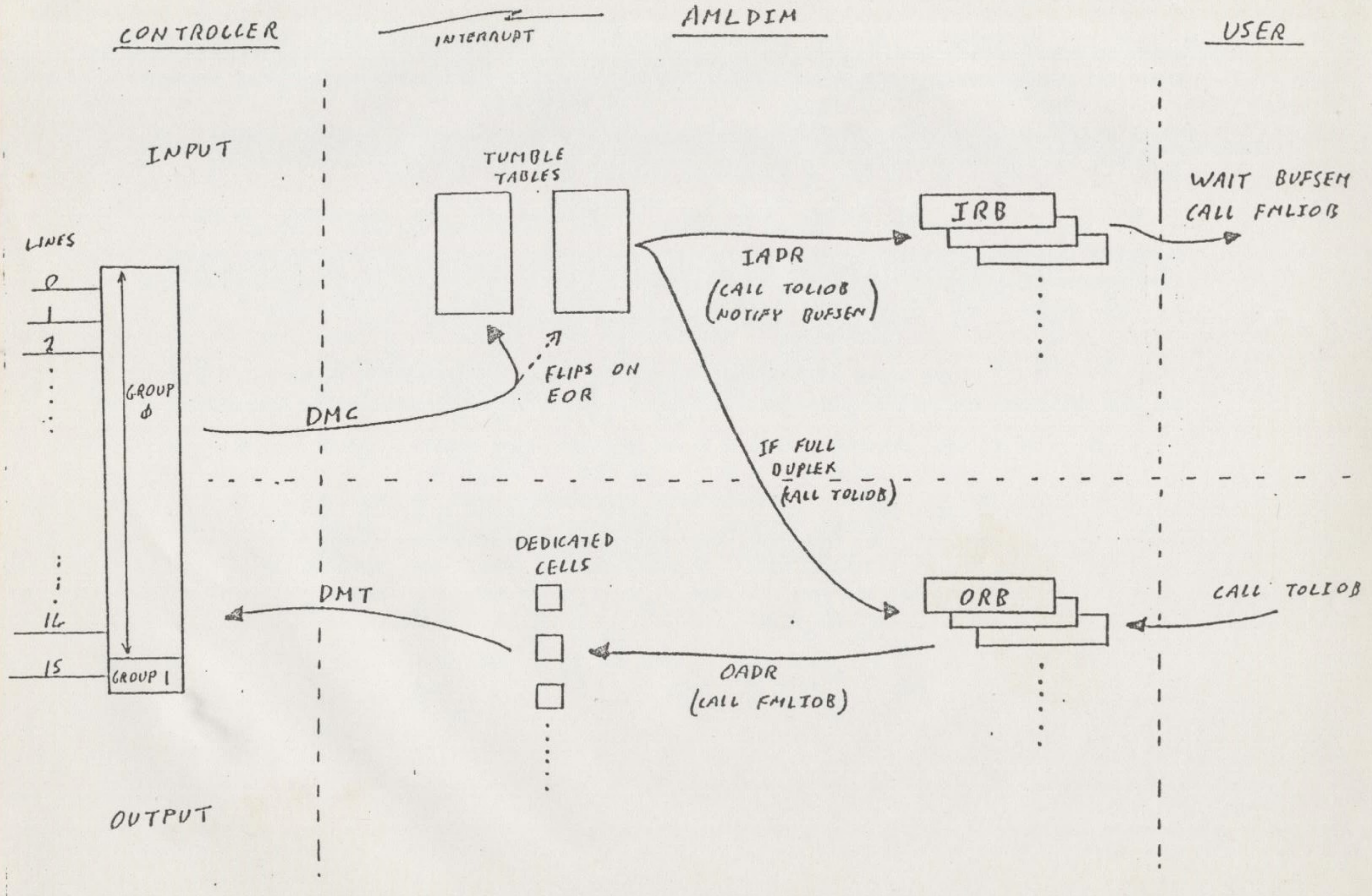
The operation performed in i) means that the AMLDIM process which, in idle state waiting on AMLSEM, gets moved onto the ready list by the dispatcher (a microcode operation). The position it occupies on the ready list is governed by its level, which is 2 for the AMLC. Only the clock and SMLC are higher. The significance of the end positioning means that if other processes were on the same level, then the AMLDIM process would be placed at the end of the chain. However, as AMLDIM is the only process at this level, this is of no significance. The level is set in the PCB at System Startup. The dispatcher then either schedules the new process (AMLDIM) if it is now at the highest level or, else continues with the current process. The latter will only occur if the current process is the clock or the SMLC.

The end result is that the AMLC gets serviced very rapidly. When the AMLDIM process has finished, then the dispatcher schedules the next process in the ready list. This could be the one that was interrupted or a higher one if another interrupt had occurred after the AMLC one.

QAMLC


CONTROLLER
INTERRUPT
AMLDIM
USER
INPUT
TUMBLE TABLES
IRB
WAIT BUFSEM
CALL FMLIOB
LINES
0
1
2
IADR
(CALL TOLIOB
NOTIFY BUFSEM)
FLIPS ON
EOR
GROUP
ϕ
DMC
IF FULL
DUPLEX
(CALL TOLIOB)
QUEUES
DMQ
ORB
CALL TOLIOB
14
15
OADR
(CALL FMLIOB)
OUTPUT

AMLC


CONTROLLER
INTERRUPT
AMLDIM
USER
INPUT
TUMBLE TABLES
WAIT BUFSEM
CALL FMLIOB
IRB
LINES
0
1
2
GROUP ∅
IADR
(CALL TOLIOB
NOTIFY BUFSEM)
FLIPS ON EOR
DMC
IF FULL DUPLEX
(CALL TOLIOB)
DEDICATED CELLS
ORB
CALL TOLIOB
DMT
16
15
GROUP 1
OADR
(CALL FMLIOB)
OUTPUT

4.3 Basic Flow Through AMLDIM

Referring to the diagram, the basic flow starts with the dispatcher (microcode) giving control to AMLDIM. After the 1st interrupt, after cold start, the process (AMLDIM) is always on a WAIT instruction. The first task is to identify the controller that interrupted. These tests are performed in Rmode because PIO cannot be performed in Vmode. Any PIO instruction is converted to an EIO which occupies 2 words. Failure to find the interrupting controller causes a HALT. Having identified the interrupting controller, the status word for that controller is input to determine what type of interrupt occurred. Three types of interrupt can occur:

i) End of Range (EOR)
ii) Character Time Interrupt (CTI)
iii) Multiple CTIs

Case i) is indicated by bit 1 being set (the sign bit)
Case ii) is indicated by bit 9. Bits 13-16 indicate the line.
Case iii) is indicated by bits 9 and 10.

Case iii) occurs if a 2nd CTI is generated before the INA instruction is issued to get the status.

If none of these cases is detected then a WAIT on AMLSEM is issued and the dispatcher reschedules another process.

Case i) EOR

Control is transferred to AMLIN. The correct tumble table is located and the table IADR is used to reference the input protocol. IADR has one entry per line which points to a protocol.

The default set up is TTYIN. The AMLC command modifies the table according to the protocol named. The subscript to point into the correct entry of IADR is obtained from the line number held in the tumble table. Control is transferred to the appropriate protocol.

There are two basic input protocols:

a)	TTYIN	Teletype input
b)	TRNSIN	Transparent input

The purpose of the protocol is to examine the incoming character and make adjustments according to the specification of the protocol. For case a) a test is made to see if it's a break character. If not then tests are made to see if XON has been enabled. The character is written to the input ring buffer using TOLIOB and if echo is required then it is also written to the output ring buffer. If the input ring buffer is full, then no attempt is made to write the character away and echo is disabled. Consequently, if the input ring buffer is not cleared, character loss results. For case b) no tests are performed except ignoring break. However, the character will not go to the input ring buffer if it is full.

Both protocols NTFY the semaphore of the line so that a user process waiting on the semaphore will be placed on the ready list.

Even though only one EOR was generated, all the tumble tables are cleared while this scan is being performed. At the end of the loop, the AMLC status is examined back at AMLDIM to see if any other interrupts had occurred (using the same status word containing EOR). If none exist th n a WAIT on AMLSEM is issued and the dispatcher gives the CPU to the next user on the ready list.

Case ii) Character Time Interrupt

On detecting a character time interrupt has occurred, a test is made to see which line caused the interrupt. If the line is the input clock line, indicated by its GFLAG being set, then extra functions are perfomred. These are:

i) Testing for loss of carry. The state indicated by a bit in the data set word word for the controller. the DTE(data terminal ready) is dropped for these lines. If carry has been dropped and DISLOG is enabled then an abort flag is set in the process abort word of the PCB. This is done at the half the clock rate (consequently usually 5 times a second). Dropping the data terminal signal for lines that have lost carry.

ii) This occurs every 3 minutes. However, problems occur with this; see section 5).
Every 3 minutes DTR is dropped for all lines that dont have carry. This caters for the case where lines that never had carry, e.g. modem lines, are accidently engaged.

iii) AMLIN is called to clear the tumble tables as for an EOR.

Then AMLOUT is used to examine all the dedicated in the current group (Ø or 1). The mechanism used to do this is to check the output ring buffer to see if any characters exist. If there are characters present then code is entered(depending on the controller type). For the DMT case, the dedicated cell is examined and if it is empty, then the OADR table is used to transfer control to the output protocol for the line. The default output protocol is TTYOUT. Others available are:

a)	TRNOUT	Transparent
b)	TRHOUT	Transparent highspeed
c)	TTHOUT	Teletype highspeed

The main difference exists between the high speed and the normal protocols. The high speed protocols use the character time interrupt bit to over-ride the slower speed of the group clock rate. If there are more than 40 characters in the output ring buffer then the CTI bit is switched on. This of course causes interrupts at the rate for the line. When there are less than 40 characters, the CTI bit is switched off and the dedicated cell is replenished at the clock rate for group zero.

In the DMQ case the queue is examined to see if it can take any more characters. Because DMQ systems do not use high speed protocol, the interrupt is caused by the last line of group zero which occurs at 110 baud.

The routine FMLIOB is used to obtain a character and place it in the dedicated cell for the line or at the bottom of the queue for DMQ.

When all the lines have been serviced, a WAIT on AMLSEM is issued.

Case iii) Multiple Character Time Interrupts

The only difference between ii) and iii) is that the AMLIN loop is executed prior to AMLOUT. This is done becuase there is no guarantee that the multiple interrupt didn't occur on the input clock line. The AMLC status word only contains the line number of the last interrupting line.

5) HANDLING SPECIAL REQUIREMENTS AND KNOWN PROBLEMS

Often it is necessary to interface special devices to the AMLC. It is important to be aware of the consequences of doing this in terms of the effect on the whole system and the effect on the device.

5.1 Known Specials

a) XON/XOFF for input devices

b) Buffered devices for output

c) Page mode devices

d) Cassette Input

e) Adding new protocols

f) Interfacing DMQ boards

a) XON/XOFF

In the standard AMLC software XON/XOFF is supported on output. This means that when the feature is enabled, sending an XOFF to PRIMOS suspends output and sending an XON resumes it. However, some devices used for input, such as cartridge devices, will respond to XON/XOFF. This is designed so that the device can transmit data at high speed with the software stopping the device when its buffers are full. The modification to PRIMOS is fairly simple and involves:

i) Testing when the tumble tables are being cleared to ensure there is enough room in the input ring buffer to hold the data.

ii) If the buffer hasn't sufficient room then placing an XOFF in the output ring buffer.

iii) Testing the state of the input ring buffer if an XOFF had been sent to see if transmission can be re-enabled.

iv) If transmission can be re-enabled, then placing an XON in the output ring buffer.

Invoking special features can be achieved by making use of spare LWORD bits. The main consideration is to ensure that extra code does not increase the overhead in AMLDIM CPU usage. Consequently test i) is the only one that needs to be placed in the interrupt loop. Test iii) can be placed in the low interrupt rate loop eg: carrier loss.

b) Buffered Devices for Output

Some output devices, such as plotters and printers, indicate when their internal buffers are full, by setting an interface line (the busy signal). The standard AMLC 5054 can detect this on pin 8 & make the state of the signal available to the software. Interfacing AMLDIM to these devices can be achieved by:

i) Incorporating a special test in AMLOUT

ii) Adding a new protocol

The modification i) is straightforward but once incorporated, gives the device to a specified line and also involves an overhead in AMLDIM, even if the device is not being used. ii) is a much more satisfactory solution as it is line independent. Care must be exercised when adding this modification that all the precautions are observed when performing the I/O required to read the AMLC status.

c) Page Mode Devices

Page mode terminals are those which transmit a whole screen of information in one burst. This causes a large quantity of information to be sent to the tumble tables. If there are a number of page mode terminals connected to the AMLC, then there is the danger that the tumble tables will not be able to handle the input rate. Consequently, loss of information will occur, which necessitates increasing the size of the tumble tables in segment Ø. The main consideration is to ensure that the disk driver still resides at location 1400. It will also be necessary to increase the size of the input ring buffers using the AMLBUF command.

d) Cassette Input

Cassette input devices are similar to page mode devices, in that they transmit burst mode packets. Consequently the size of the input ring buffers will need to be increased and the tumble tables may need to be increased. If the device responds to XON/XOFF, then the considerations in a) need to be borne in mind.

e) Adding new Protocols

Adding new protocols is a fairly straightforward process. The tables in NLKCOM will need to be adjusted to reference the new protocol name (as input with the AMLC command) to the driver name in AMLDIM. The new protocol code will need to be added to AMLDIM using the logic contained in the existing protocols ie: use of TDLIOB and FMLIOB to manipulate the characters. The only other important consideration is to ensure that the generated code doesn't overflow the page boundries set up in MAPGEN.

f) Interfacing DMQ boards

Adding DMQ boards to the standard system causes no difficulty. The problem comes when a special addition has to be incorporated. The DMQ only affects specials that require suspension of output based on certain requirements. The length of the queue must be taken into account because suspension of transfer from the ring buffer to the queue doesn't affect the DMQ going from queue to the AMLC. It is therefore necessary to pack out the queue with null characters which don't get sent to the device.

5.2 Known Problems

Certain known problemsexist which can be got round by using certain techniques.

If forced logout on disconnect is configured (in the CONFIG file) direct connect devices may be logged out. The object is to drop DTR (Data Terminal Ready) on lines with no carrier. However this is done by pretending all lines have carrier. Any line that never had carrier (ie: a direct connected line) will be force logged out. The solution for devices that generate DTR is to use cable type 1470. For devices that do not generate DTR strap DTR from the AMLC to carrier. For the system console being operated as a USRASR terminal, the carrier must appear high on the line that corresponds to the buffer being switched. The alternative is to set the LWORD to zero.

If forced logout on disconnect is enabled, then output may not be turned on. This is because the logout message is attempted before the LWORD is changed to allow output (ie: the buffer number inserted). If the output ring buffer is full then the process (user) hanges on a semaphore. Message all now can cause the ring buffer to fill.

Unstable carrier can cause problems such as random disconnects.

Problems can occur with UK Modems because noise on the line may cause the modem to think carrier is permanently high. Carrier high with no one logged in can cause a modem to become permanently engaged by a wrong number.

The maximum size of all ring buffers (in total) must be less than 32K words.

6) P300 DIFFERENCES

The mechanisms used by the AMLC hardware are independent of system as the same controller is used throughout. The main difference between the P300 and P400 concerns the segmented architecture of the latter.

The AMLC driver AMLDIM doesn't differ significantly between the P300 and P400. The technique of tumble tables, dedicated cells and ring buffers applies. DMQ is not available on the P300.

The most important difference concerns the way the code is entered. As there is no process exchange mechanism, the interrupt address is the entry point for AMLDIM. The DMX memory areas exist in the same segment as the driver. The ring buffers exist in a pseudo segment which is addressed through the memory mapping tables.

The parameters of the AMLC software are fixed and changes can only be made at source level. The most common change is the buffer size. This can be achieved by modifying the module TFLIOB. The main consideration is to ensure that the centronics buffer start address is located on a page boundary.

The suspension of users is achieved by a state vector. This means that if a user requires input, he will not get access to the ring buffer until a time slice interval (unlike PRIMOS IV) where he will be waiting on BUFSEM and get put on the ready list by AMLDIM. This of course has consequances when servicing fast devices.

XON/XOFF is not implemented in the standard system, although insertion of the code is fairly straightforward.

AMLDIM ENHANCEMENTS

. BUFFERED PROTOCOL (REVERSE CHANNEL)

- LWORD BIT-5 SET-DETECT BUSY

  BIT-6 (USED ONLY IF BIT-5 SET)

  ON - IF DATA SET SENSE HIGH ISSUE XOFF, ELSE XON

  OFF - IF DSS LOW ISSUE XOFF, ELSE XON

. TRANSMIT DISABLED WHEN BUFFER EMPTY 5 SECONDS.

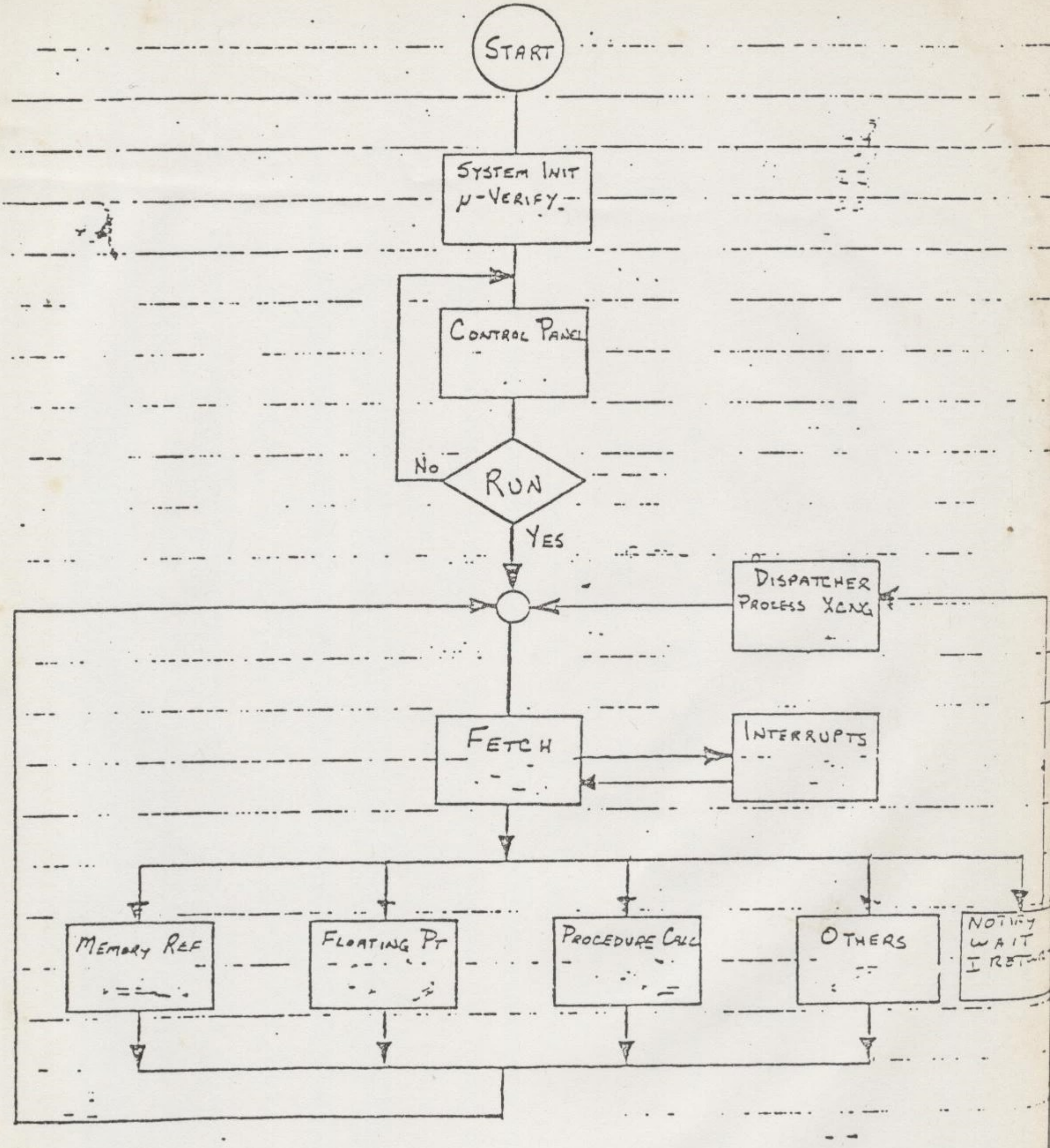
. DTRDRP CONFIG DIRECTIVE AND DROPDTR COMMAND.

. BUFFER OVERFLOW DETECTED USING NAK ('225) CHARACTER.

IF ONLY ONE CHARACTER SPACE REMAINS IN THE INPUT RING BUFFER, A NAK WILL BE PLACED THERE. A SUBSYSTEM CAN CHECK FOR THIS AND REQUEST A RETRANSMIT AFTER ISSUING A CALL TO BUFCLR.

. PARITY ERROR DETECTION

IF BIT 8 OF THE LWORD IS SET, AMLDIM WILL REPLACE ALL PARITY ERRORS WITH A NAK CHARACTER. THESE MAY BE HANDLED AS FOR BUFFER OVERFLOWS.

START
SYSTEM INIT
μ-VERIFY
CONTROL PANEL
No
RUN
YES
DISPATCHER
PROCESS XCNG
FETCH
INTERRUPTS
MEMORY REF
FLOATING PT
PROCEDURE CALL
OTHERS
NOTIFY
WAIT
I RET

V-Mode Register Description:

SCRATCH			DMX			CURRENT REGISTER SET (CRS)			
RSØ			RS1			RS2	RS3		
ADR	HIGH	LOW	ADR	HIGH	LOW	ADR	ADR	HIGH	LOW
Ø	TRØ	-	4Ø	-	-	1ØØ	14Ø	GRØ:OLT2	-
1	TR1	-	41	-	-	1Ø1	141	GR1:PTS	-
2	TR2	-	42	-	-	1Ø2	142	GR2(1,A,LH)	(2,B,LL)
3	TR3	-	43	-	-	1Ø3	143	GR3(EH)	(EL)
4	TR4	-	44	-	-	1Ø4	144	GR4	-
5	TR5	-	45	-	-	1Ø5	145	GR5(3,S,Y)	-
6	TR6	-	46	-	-	1Ø6	146	GR6	-
7	TR7	-	47	-	-	1Ø7	147	GR7(Ø,X)	-
1Ø	RDMX1	-	5Ø	-	-	11Ø	15Ø	FAR1(13)	-
11	RDMX2	-	51	-	-	111	151	FLR1	-
12	-	RATMPL	52	-	-	112	152	FAR2(4)	(5)
13	RSGT1	-	53	-	-	113	153	FLR2:VSC(6)	-
14	RSGT2	-	54	-	-	114	154	PB	-
15	RECC1	-	55	-	-	115	155	SB(14)	(15)
16	RECC2	-	56	-	-	116	156	LB(16)	(17)
17	-	REOIV	57	-	-	117	157	XB	-
2Ø	ZERO	ONE	6Ø	(2Ø)	(21)	12Ø	16Ø	DTAR3(1Ø)	-
21	PBSAVE	-	61	-	-	121	161	DTAR2	-
22	RDMX3	-	62	(22)	(23)	122	162	DTAR1	1
23	RDMX4	-	63	-	-	123	163	DTARØ	-
24	C377	-	64	(24)	(25)	124	164	KEYS	(MODALS)
25	-	-	65	-	-	125	165	OWNER	-
26	-	-	66	(26)	(27)	126	166	FCODE(11)	-
27	-	-	67	-	-	127	167	FADDR	(12)
3Ø	PSWPB	-	7Ø	(3Ø)	(31)	13Ø	17Ø	TIMER	-
31	PSWKEYS	1	71	-	-	131	171	-	-
32	PPA:PLA	PCBA	72	(32)	(33)	132	172	-	-
33	PPB:PLB	PCBB	73	-	-	133	173	-	-
34	DSWRMA	-	74	(34)	(35)	134	174	-	-
35	DSWSTAT	-	75	-	-	135	175	-	-
36	DSWPB	-	76	(36)	(37)	136	176	-	-
37	RSAVPTR	-	77	-	-	137	177	-	-

NOTICE - Numbers in parentheses ( ) show P3ØØ Address Mapping

Definitions

TR Temporary Registers
TR7 - Saved return pointer on a crash (automatic save)

RDMX Register DMX
RDMX1 - Used by DMC, buffer start pointer
RDMX2 - REA at time of DMX trap
RDMX3 - Save RD during DMQ
RDMX4 - Used as working register

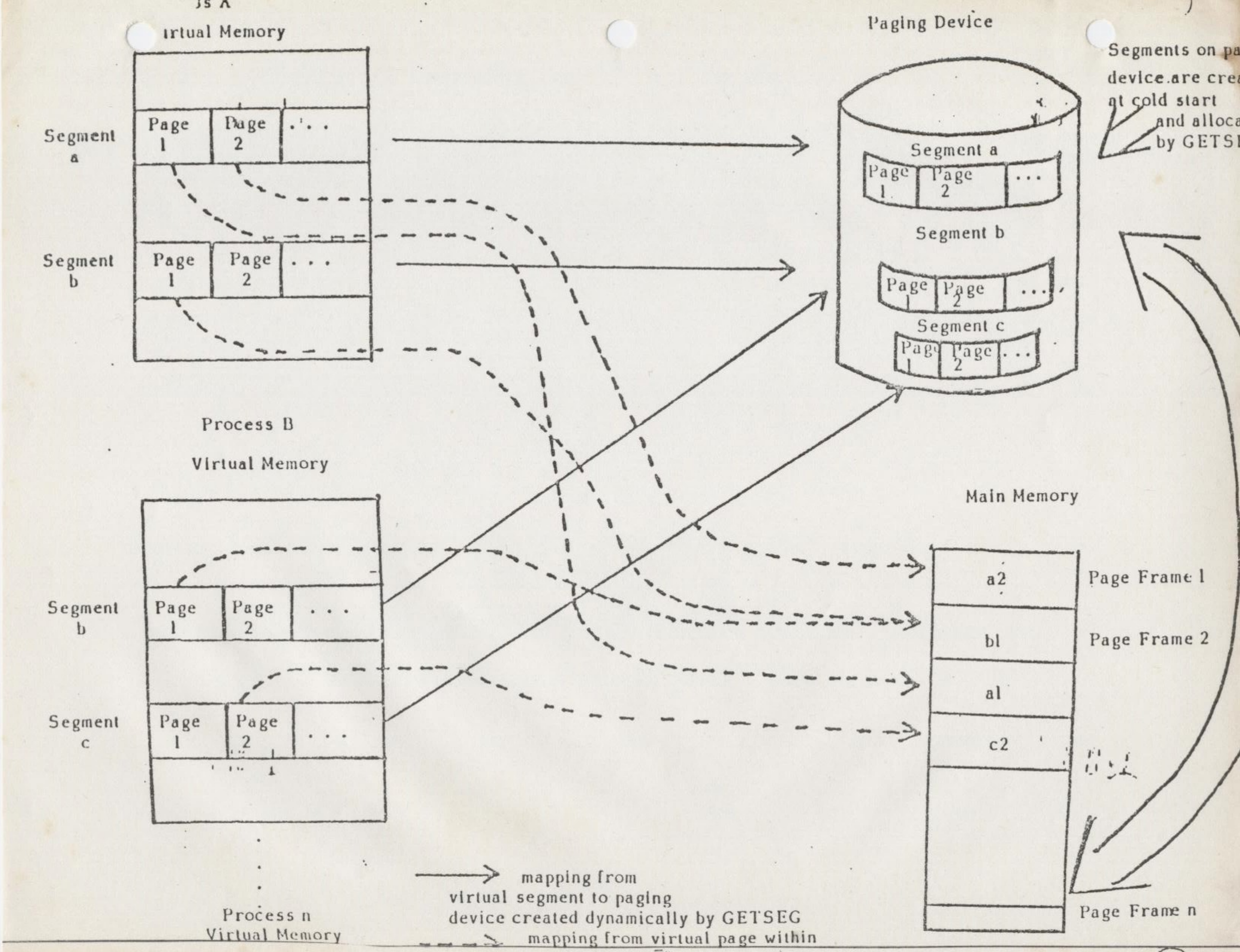
RATMPL Read Address Trap Map to rP Low

RSGT Register Segmentation Trap
RSGT1 - SDW2 / address of Page Map
RSGT2 - contents of Page Map / SDW2

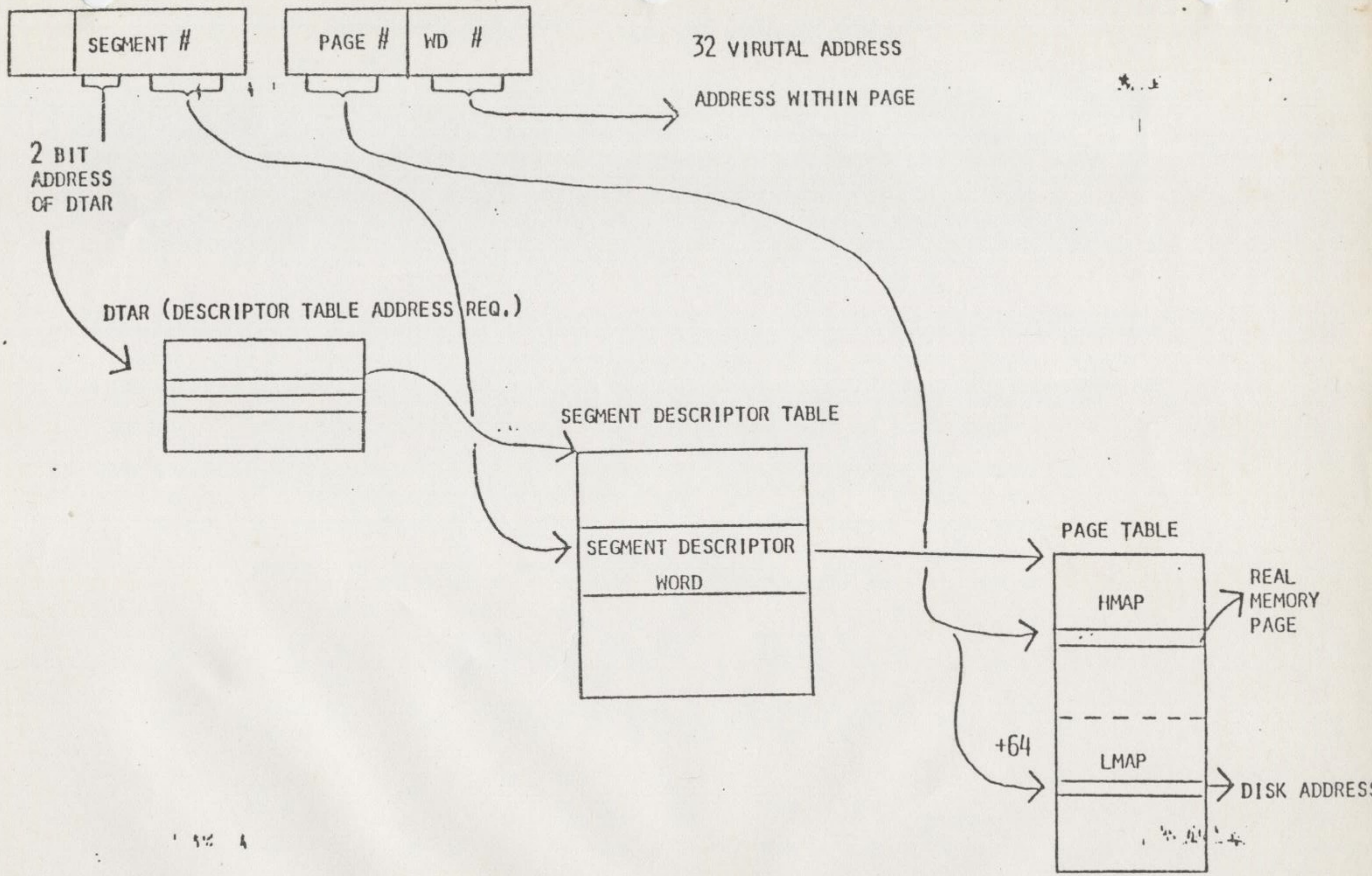
REOIV	Register End of Instruction Vector
ZERO/ONE	Constants
PBSAVE	Procedure Base SAVE saved return pointer when return pointer used elsewhere
C377	Constant
PSWPB	Processor Status Word Procedure Base return pointer for interupt return (also used for Prime 300 compatibility)
PSWKEYS	Processor Status Word KEYS KEYS for interupt return (also used for Prime 300 compatibility)
PPA	Pointer to Process A
PLA	Pointer to Level A
PCBA	Process Control Block A
PPB	Pointer to Process B
PLA	Pointer to Level B
PCBB	Process Control Block B
DSWRMA	Diagnostic Status Word RMA RMA at last Check Trap
DSWSTAT	Diagnostic Status Word STATus
DSWPB	Diagnostic Status Word Procedure Base Return pointer or PBSAVE at last check
RSAVPTR	Register SAVE Pointer Location of Register Save Area after Halt
GR	General Register
OLT2	Old Length and Type
PTS	Pointer To Sign
FAR1	Field Address Register 1
FLR1	Field Length Register 1
FAR2	Field Address Register 2
FLR2	Field Length Register 2
PB	Procedure Base PBH - RPH PBL - Ø
SB	Stack Base
LB	Link Base
XB	Temporary (auxiliary) base
DTAR	Descriptor Table address registers
KEYS	See below
MODALS	See below
OWNER	Pointer to PCB of process owning this register set
FCODE	Fault CODE
FADDR	Fault ADDRess
TIMER	1-millisecond process timer (used for time-slice)

V-Mode Register Usage:

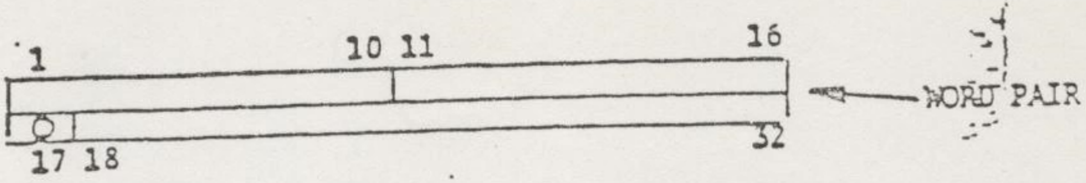
STLR/	Address	
LDLR	Trap	Usage
–	7	P (program counter)
2 H	1	A (accumulator, high half of L)
2 L	2	B (double-precision, low half of L)
3 H,L	–	EH,EL (accumulator extension for MPL DVL)
5 H	3	Y (alternate index), S (stack)
7 H	0	X (index)
10 H	13	–
10,11	–	(field address and length register 0)
12,13	–	(field address and length register 1)
12 H	4	(floating accumulator, mantissa high)
12 L	5	(mantissa middle)
13 H	6	(exponent)
13 L	–	(mantissa low, double-precision)
14 H,L	–	PB (procedure base)
15 H,L	14,15	SB (stack base)
16 H,L	16,17	LB (linkage base)
17 H,L	–	XB (temporary base)
20 H	10	(high half of DTAR3)
20 H,L	–	DTAR3 (descriptor table address, segments 3072-4095)
21 H,L	–	DTAR2 (segments 2048-3071)
22 H,L	–	DTAR1 (segments 1024-2047)
23 H,L	–	DTAR0 (segments 0-1023)
24 H,L	–	keys, modals
25 H,L	–	OWNER (address of process control block of process owning register contents)
26 H	11	FCODE (fault code)
27 H,L	–	FADDR (fault address)
27 L	12	(fault address word number)
30 H	–	process 1024-microsecond c.p.u timer



ss A
irtual Memory
Segment a
Page 1
Page 2
. . .
Segment b
Page 1
Page 2
. . .
Process B
Virtual Memory
Segment b
Page 1
Page 2
. . .
Segment c
Page 1
Page 2
. . .
Process n
Virtual Memory
Paging Device
Segment a
Page 1
Page 2
. . .
Segment b
Page 1
Page 2
. . .
Segment c
Page 1
Page 2
. . .
Segments on pa
device are cre
at cold start
and alloca
by GETS
Main Memory
a2
b1
a1
c2
Page Frame 1
Page Frame 2
Page Frame n
mapping from
virtual segment to paging
device created dynamically by GETSEG
mapping from virtual page within

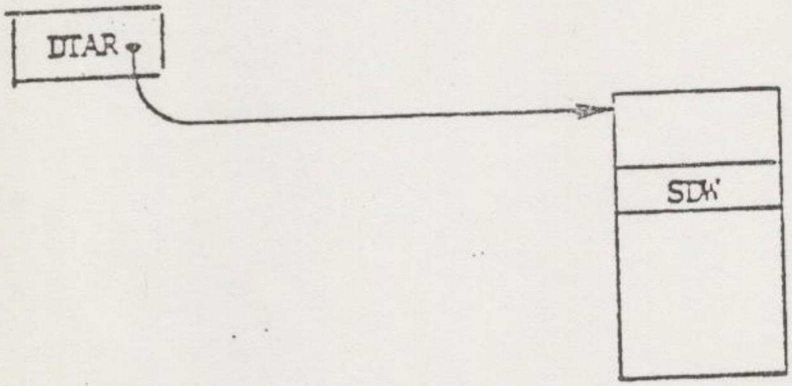

SEGMENT #
PAGE #
WD #
32 VIRUTAL ADDRESS
ADDRESS WITHIN PAGE
2 BIT
ADDRESS
OF DTAR
DTAR (DESCRIPTOR TABLE ADDRESS REQ.)
SEGMENT DESCRIPTOR TABLE
SEGMENT DESCRIPTOR
WORD
PAGE TABLE
HMAP
REAL
MEMORY
PAGE
+64
LMAP
DISK ADDRESS

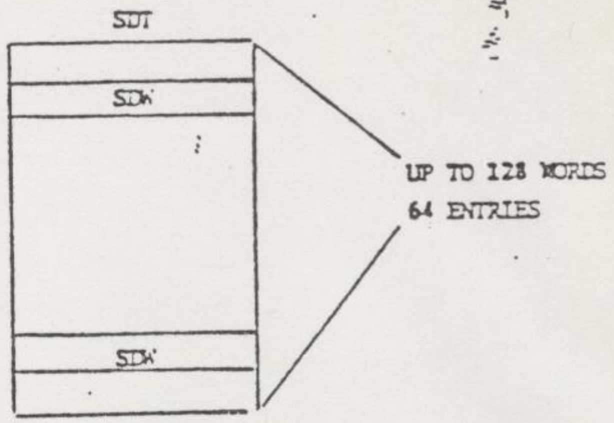
DTAR



1-10 - # OF ENTRIES IN SDT

11 - 32 - HIGH ORDER 21 BITS OF PHYSICAL ADDRESS (LOW ORDER BIT TAKEN AS ZERO SINCE IT ALWAYS ACCESSES A WORD PAIR IN SDW.





1			10 11	16
PMADR -HI			MBZ	
F	AAA	BBB	CCC	PMADR -LOW
17	18-20	21-23	24-26	27 32

BITS 1-10!27-32 = PHYSICAL ADDRESS OF PAGE MAP. (MUST BE ON A 64K BOUNDARY.

BITS 18-20 = SPECIFY THE RING RIGHTS FOR RING 1

BITS 21-23 = RESERVED FOR FUTURE (Ring 2 rights)

BITS 24-26 = SPECIFY THE RING RIGHTS FOR RING 3

NOTE: RING 0 ALWAYS HAS ALL ACCESS RIGHTS.

HMAP ENTRY

1	2	3	4	5 16
V	R	U	S	PHYSICAL PAGE #

V - VALID = PAGE IS IN MEMORY

R - REFERENCED = PAGE WAS REFERENCED

U - UN-MODIFIED = IF THE PAGE HAS BEEN MODIFIED,THIS BIT IS 0

S - SHARED BIT = RESERVED FOR FUTURE MULTI-PROCESSOR SHARING ! probably used by P750-AP (P850)

BITS 5 - 16 = 12 BIT PHYSICAL PAGE #

LMAP ENTRY

1 2	3	4	5 16
LK #	NO OLD	ALT PDV	DISK INDEX TO 8 PAGES

LOCK # = IF 0,PAGE NOT LOCKED

NO OLD = NO OLD COPY EXISTS ON DISK,IF BIT SET

ALT = USE ALTERNATE PAGING DEVICE

BITS 5 - 16 = DISK TRACK ADDRESS (INDEX TO 8 PAGES)

SEGMENT SHARING

- DTARs 0 and 1 are shared by all processes. They are not altered on a process exchange.

  Thus all processes hsare the same segments numbered 0...3777 (octal).

- Each user has his own private settings for DTARs 2 and 3 stored in his Process Control Block. These settings are swapped on a process exchange.

  Thus each user can have his own individual segments numbered 4000...7777(octal).

- But segments in DTARs 2 and 3 can be shared too. This happens when two (or more) users have segment descriptors pointing to the same page table.

  This form of sharing need not be system-wide, and the segment number assigned to the shared segment need not be identical in all processes.

  This type of sharing is not allowed under current release of PRIMOS.

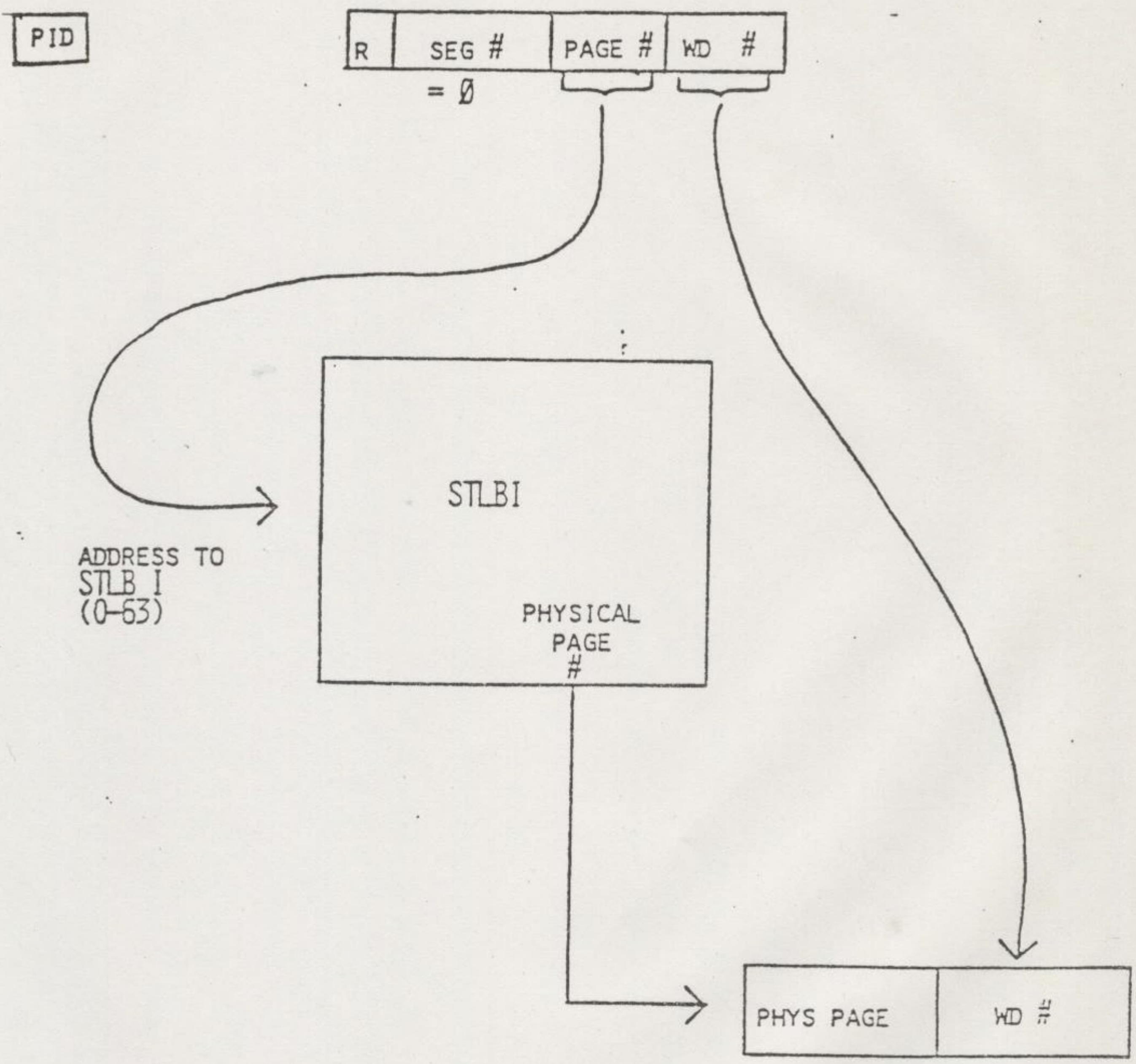
	OPERATING SYSTEM	USER APPLICATIONS
SHARED	DTAR 0 (0...17777) operating system code	DTAR 1 (2000...3777) shared editor shared libraries
NONSHARED	DTAR 3 (6000...7777) per-user system tables	DTAR 2 (4000...5777) normal user code

TYPICAL DTAR USAGE

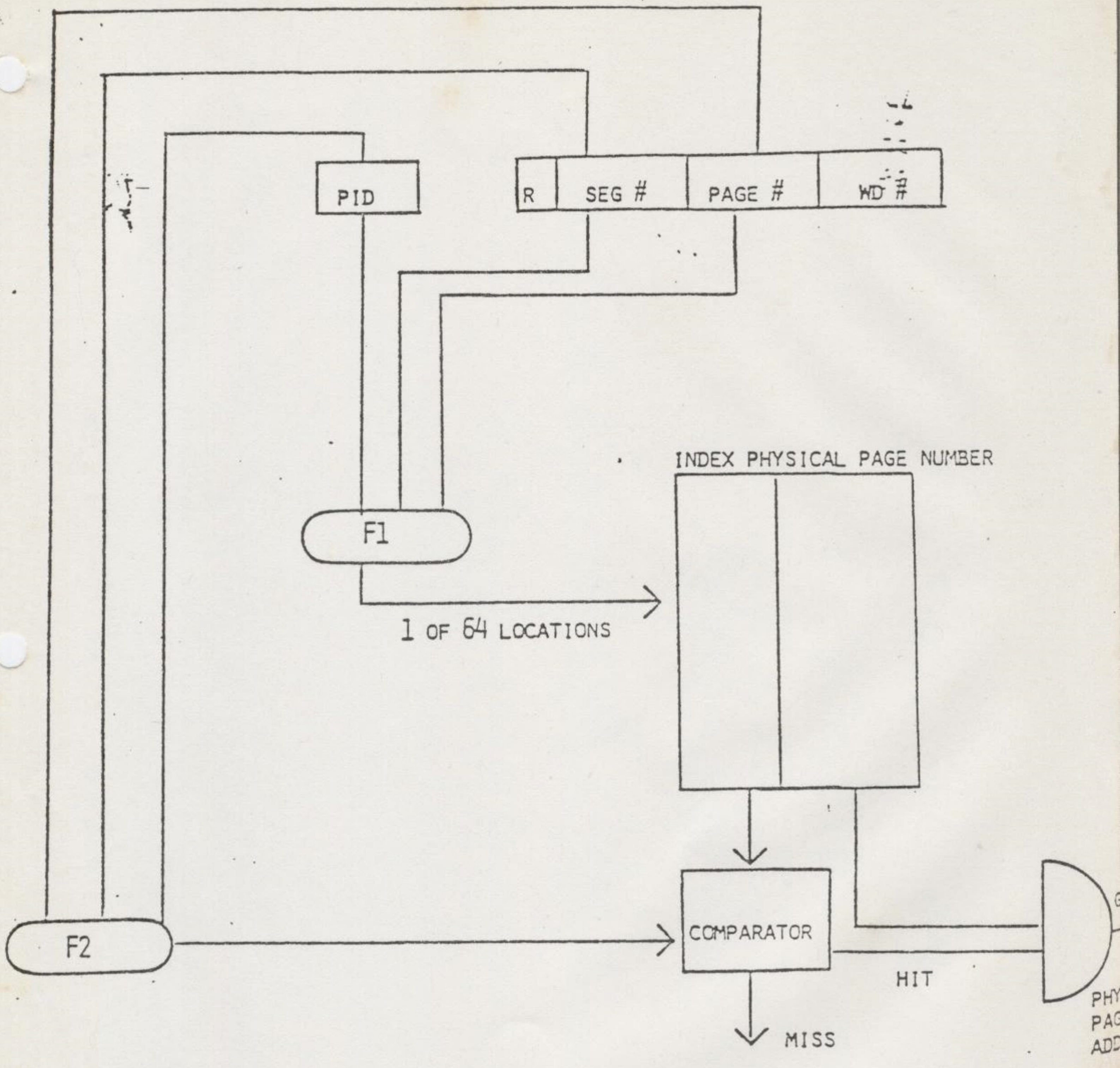
STLB I (IOTLB)

SIMPLIFIED DATA FLOW

(SEGMENT # = 0)



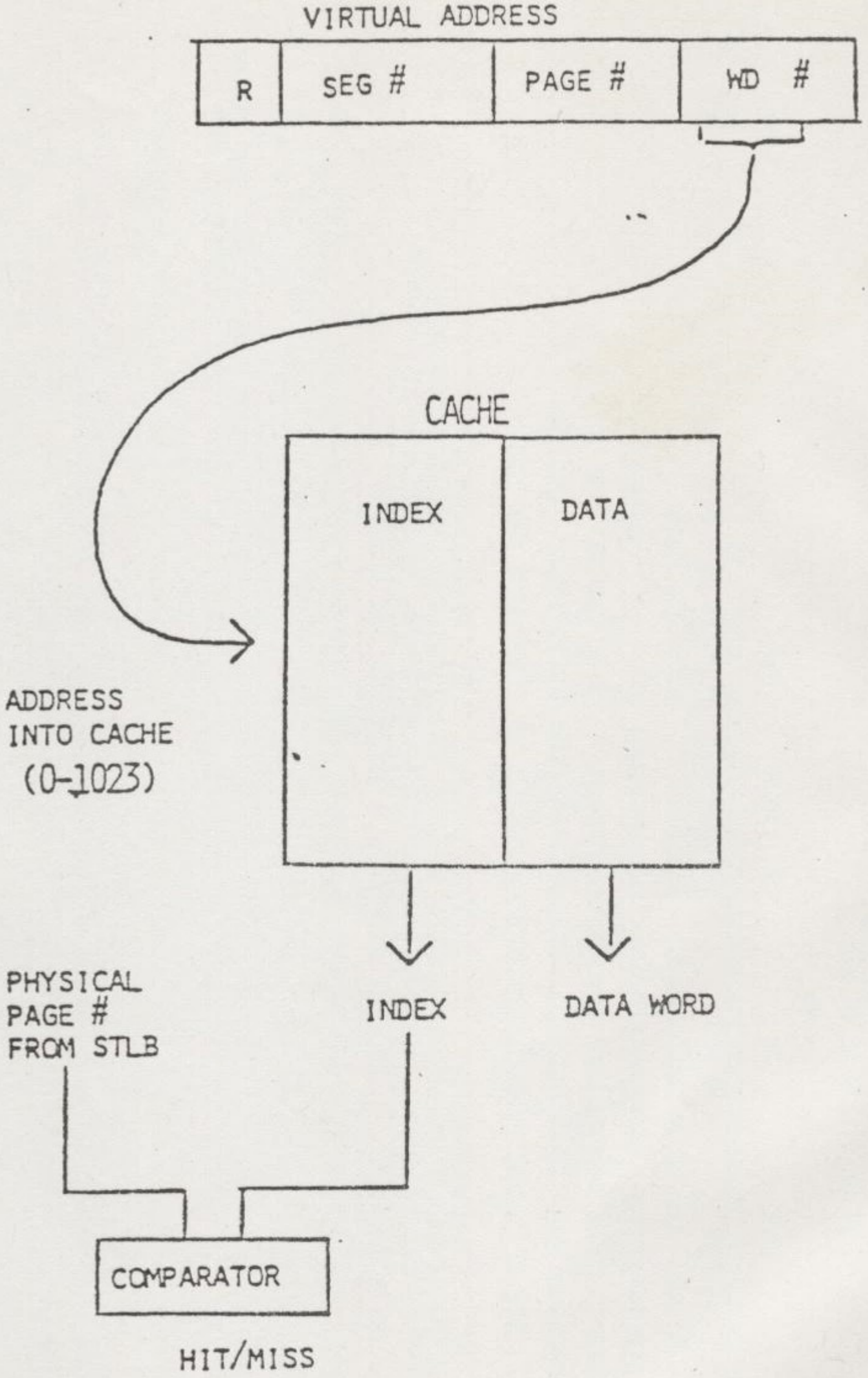
STLB II SIMPLIFIED DATA FLOW
(SEGMENT # ≠ Ø)

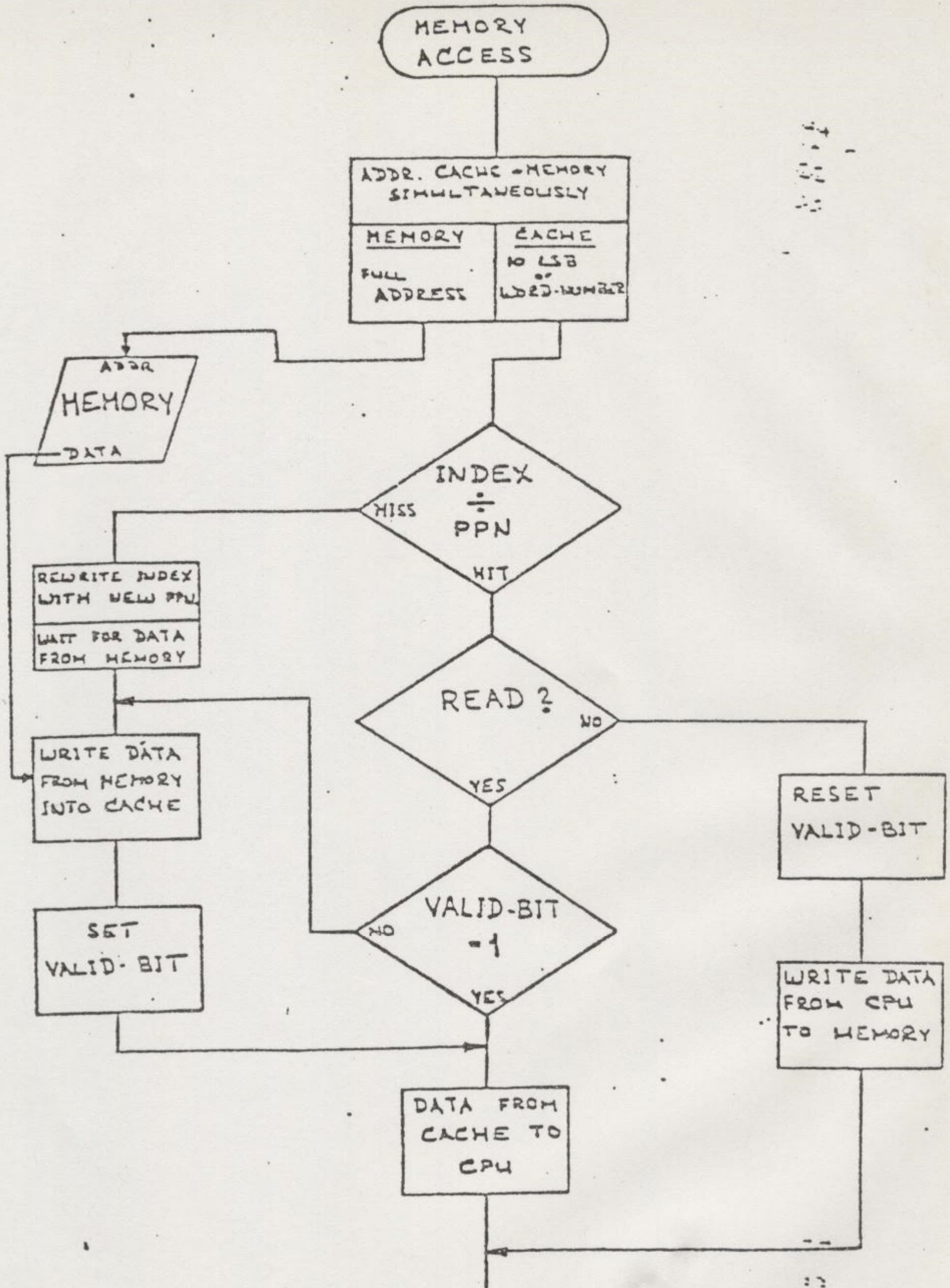


NOTE 1: F1 AND F2 ARE HASH FUNCTIONS

NOTE 2: IF MISS EXISTS, MAPPING FUNCTION IS PERFORMED ALONG WITH HAS FUNCTION F2. PHYSICAL PAGE NUMBER PLUS HAS F2 ARE WRITTEN INTO STLB.

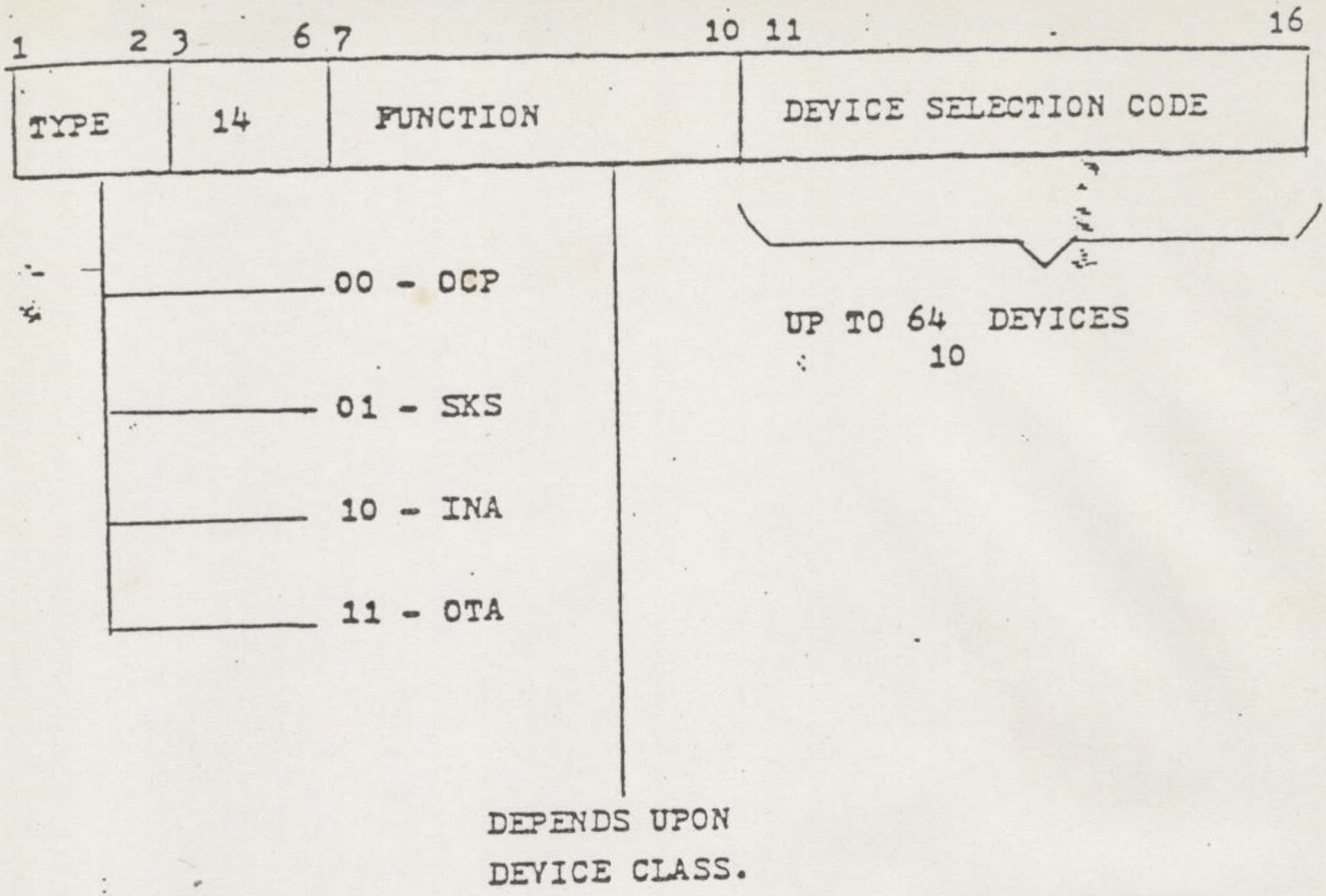
CACHE (P350 - P650)



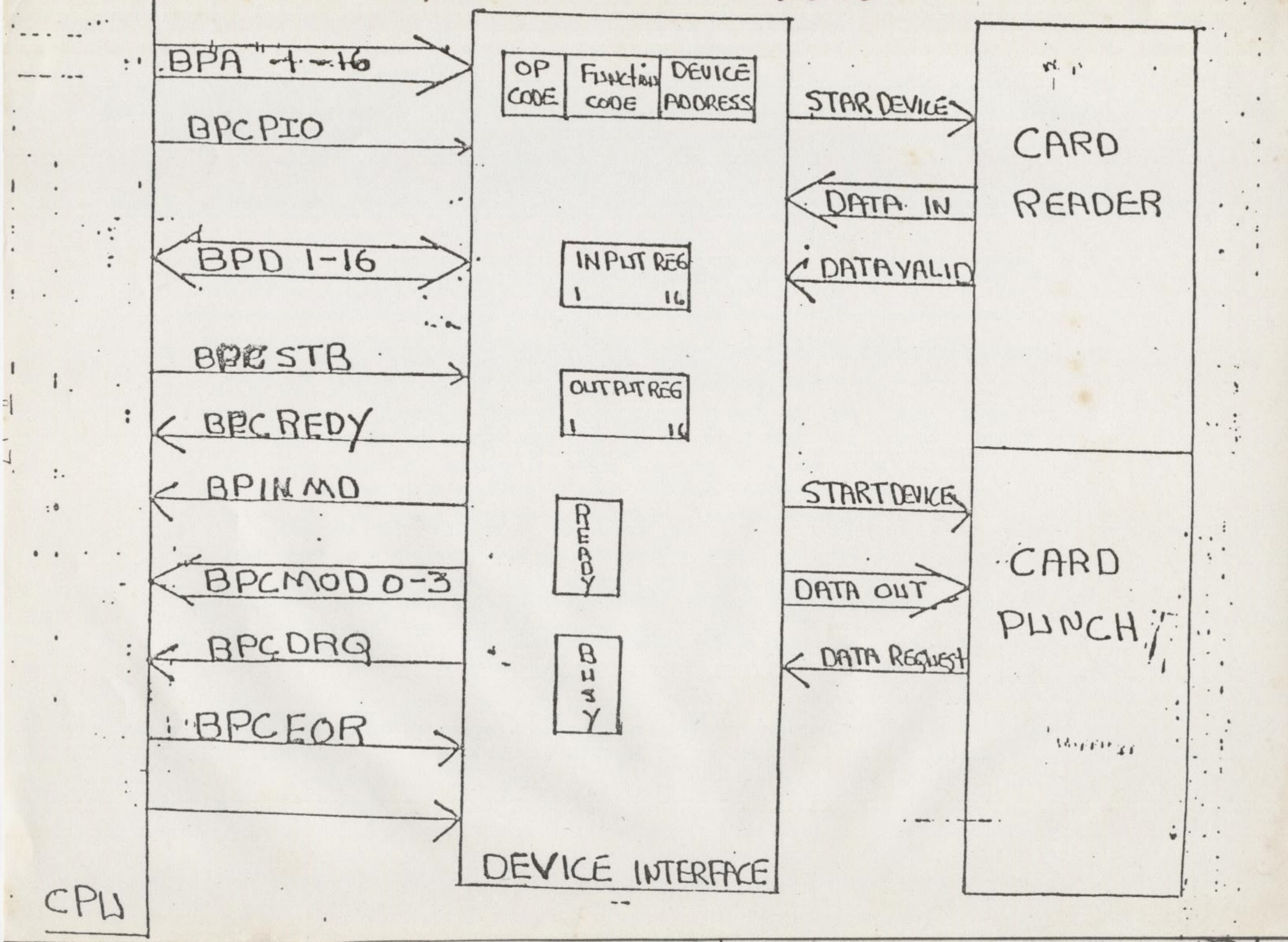

MEMORY ACCESS
ADDR. CACHE + MEMORY SIMULTANEOUSLY
MEMORY
FULL ADDRESS
CACHE
10 LSB or WORD-NUMBER
ADDR
MEMORY
DATA
INDEX ÷ PPN
MISS
HIT
REWRITE INDEX WITH NEW PPN
WAIT FOR DATA FROM MEMORY
READ ?
NO
YES
WRITE DATA FROM MEMORY INTO CACHE
VALID-BIT = 1
NO
YES
SET VALID-BIT
RESET VALID-BIT
WRITE DATA FROM CPU TO MEMORY
DATA FROM CACHE TO CPU

3-1-78 ф

* INPUT - OUTPUT *



I/O DATA FLOW


BPA 1-16
BPCPIO
BPD 1-16
BPC STB
BPCREDY
BPINMD
BPCMOD 0-3
BPCDRQ
BPCEOR
CPU
OP CODE
Function CODE
DEVICE ADDRESS
INPUT REG
1
16
OUTPUT REG
1
16
READY
BUSY
DEVICE INTERFACE
STAR DEVICE
DATA IN
DATA VALID
START DEVICE
DATA OUT
DATA REQUEST
CARD READER
CARD PUNCH

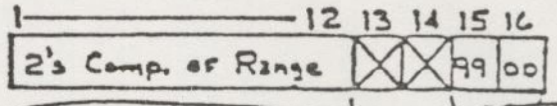
DMx Operation

Description:

A. DMT - Direct memory transfer; controller supplies memory Addres
directly; Fastest of all DMx

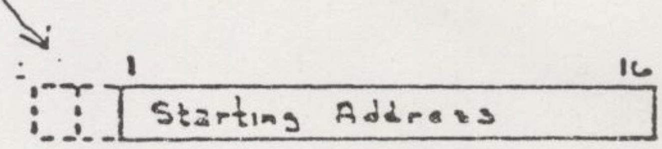
B. DMA - Direct memory access; controller supplies "channel number
to CPU; CPU accesses a pair of locations in Registe
File which will supply RANGE and STARTING LOCATION
for transfer; 8 channels of DMA; slower than DMT
Reg. Files locations 20—37 reserved for DMA

1st Location



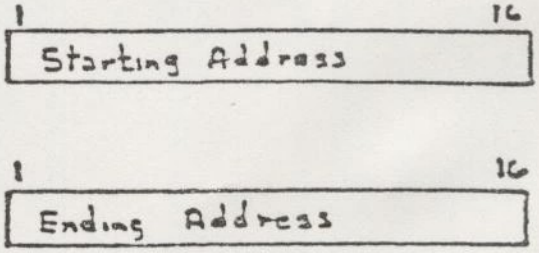
Range = Number of Words to be transferred (4K
99 00 = High order address bits to allow
transfers anywhere within 256K

2nd Location



Bits 15 & 16 of 1st Location extend address to

C. DMC - Direct Memory Channel; controller supplies "channel number" t
CPU; CPU access a pair of Memory Locations (adja
which supply STARTING ADDRESS and ENDING ADD
≈32K ~~1000~~ channels ~~(40 to 3778)~~; slowest of all DMx;
is max. range



Limited by controller

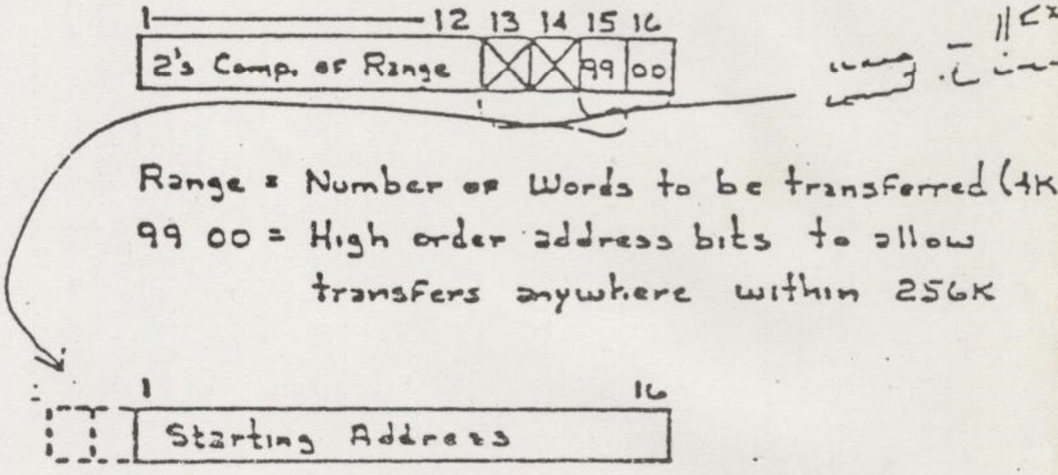
DMx Operation

Description:

A. DMT - Direct memory transfer; controller supplies memory Address directly; Fastest of all DMx

B. DMA - Direct memory access; controller supplies "channel number" to CPU; CPU accesses a pair of locations in Register File which will supply RANGE and STARTING LOCATION for transfer; 8 channels of DMA; slower than DMT Reg. Files locations 20—37 reserved for DMA

1st Location



Range = Number of Words to be transferred (4K

99 00 = High order address bits to allow transfers anywhere within 256K

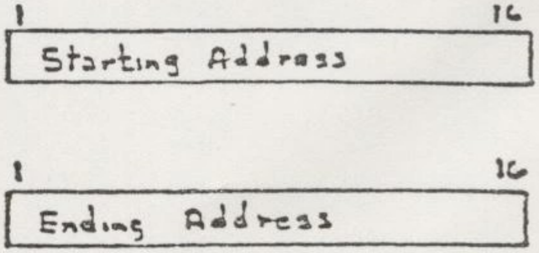
2nd Location

Bits 15&16 of 1st Location extend address to

C. DMC - Direct Memory Channel; controller supplies "channel number" to CPU; CPU access a pair of Memory Locations (adja which supply STARTING ADDRESS and ENDING ADD ~~1000~~ channels ~~(40 to 3777s~~); slowest of all DMx;

≈32K

is max. range

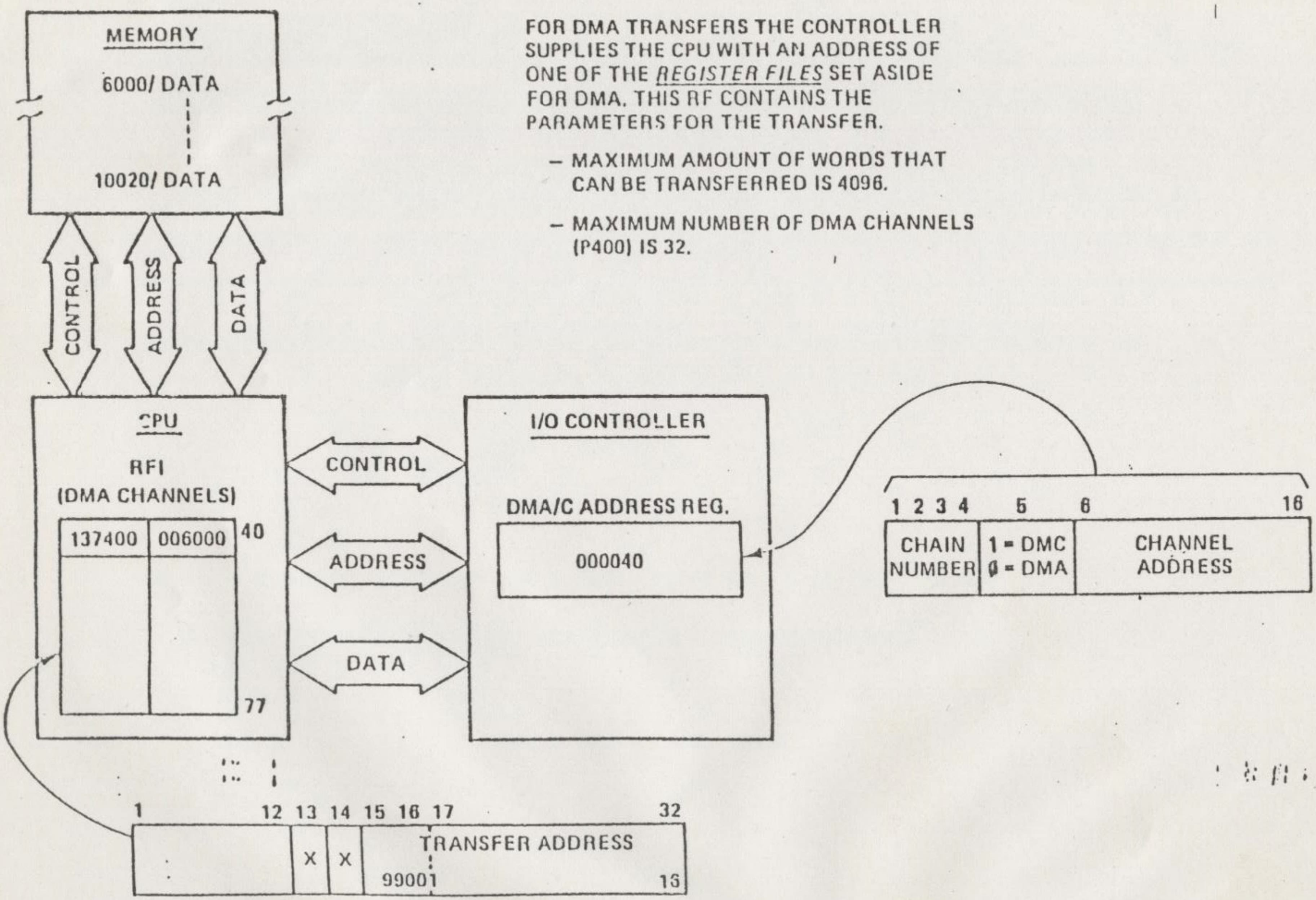


Limited by controller

DIRECT MEMORY ACCESS (DMA)*

FOR DMA TRANSFERS THE CONTROLLER SUPPLIES THE CPU WITH AN ADDRESS OF ONE OF THE *REGISTER FILES* SET ASIDE FOR DMA. THIS RF CONTAINS THE PARAMETERS FOR THE TRANSFER.

- MAXIMUM AMOUNT OF WORDS THAT CAN BE TRANSFERRED IS 4096.
- MAXIMUM NUMBER OF DMA CHANNELS (P400) IS 32.

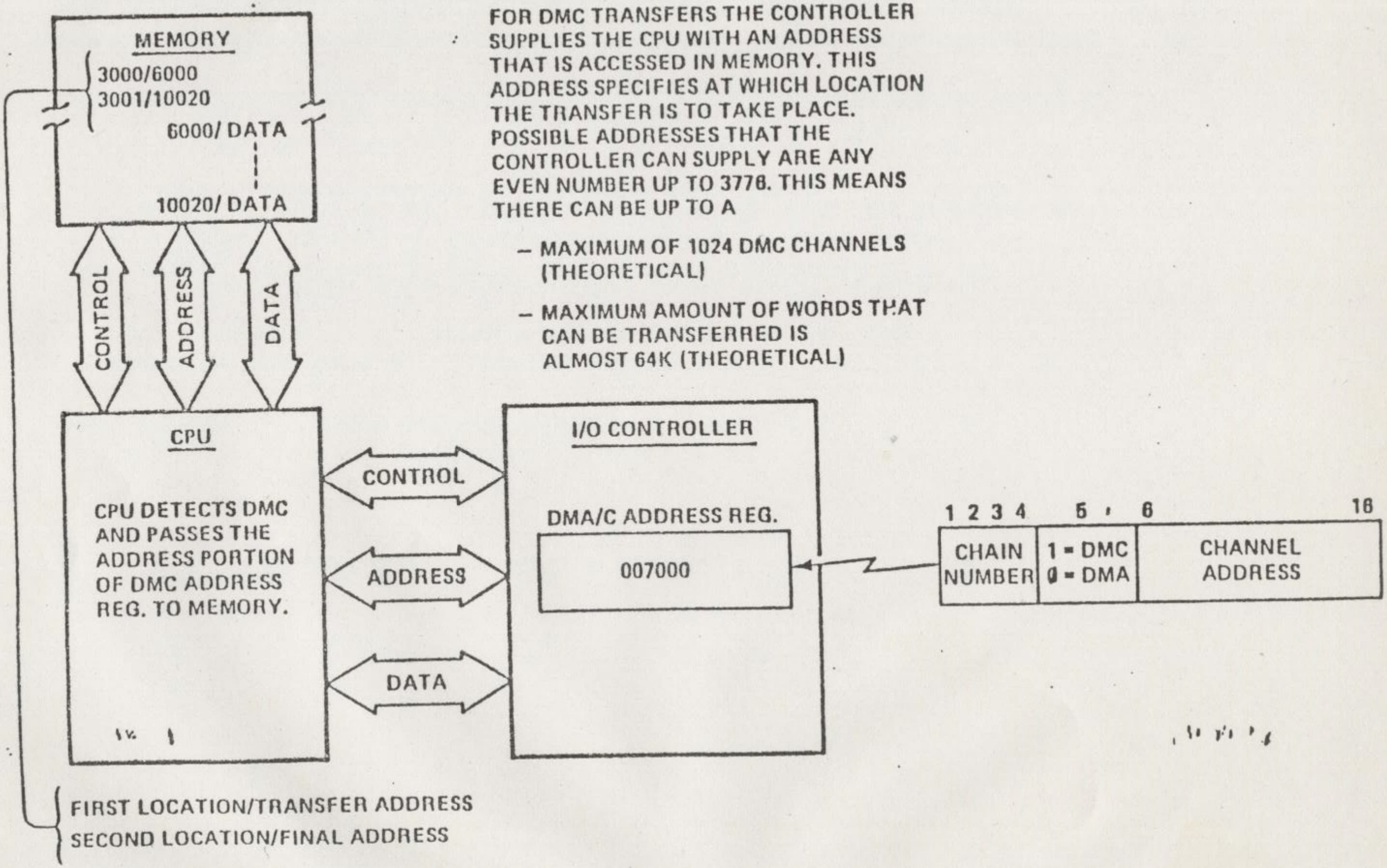


* *Example shows parameters for a 1040 word transfer from/to locations 6000–10020.*

DIRECT MEMORY CHANNEL (DMC)*

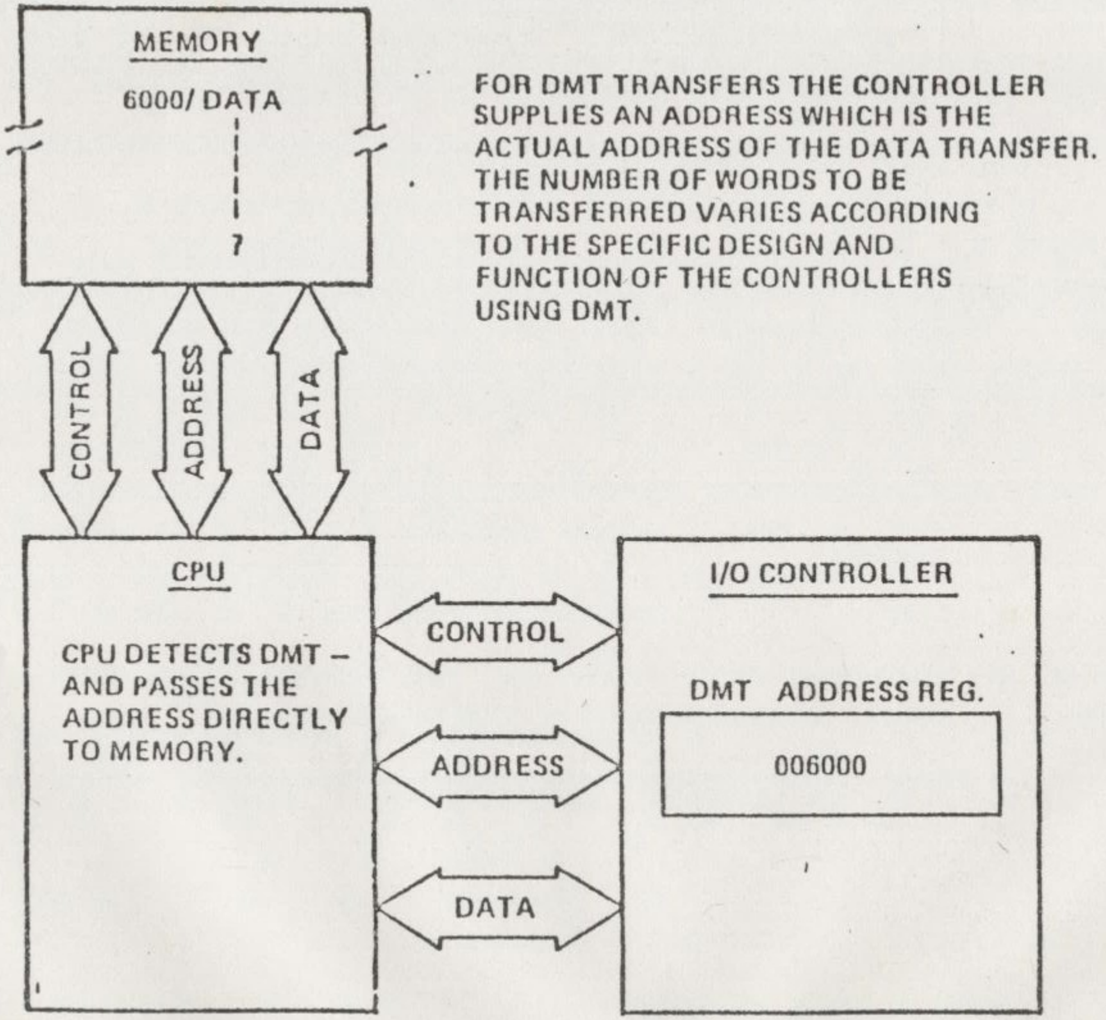
FOR DMC TRANSFERS THE CONTROLLER SUPPLIES THE CPU WITH AN ADDRESS THAT IS ACCESSED IN MEMORY. THIS ADDRESS SPECIFIES AT WHICH LOCATION THE TRANSFER IS TO TAKE PLACE. POSSIBLE ADDRESSES THAT THE CONTROLLER CAN SUPPLY ARE ANY EVEN NUMBER UP TO 3776. THIS MEANS THERE CAN BE UP TO A

- MAXIMUM OF 1024 DMC CHANNELS (THEORETICAL)
- MAXIMUM AMOUNT OF WORDS THAT CAN BE TRANSFERRED IS ALMOST 64K (THEORETICAL)



* *Example shows parameters for a 1040 word transfer from/to locations 6000–10020.*

DIRECT MEMORY TRANSFER (DMT)*



FOR DMT TRANSFERS THE CONTROLLER SUPPLIES AN ADDRESS WHICH IS THE ACTUAL ADDRESS OF THE DATA TRANSFER. THE NUMBER OF WORDS TO BE TRANSFERRED VARIES ACCORDING TO THE SPECIFIC DESIGN AND FUNCTION OF THE CONTROLLERS USING DMT.

CPU DETECTS DMT – AND PASSES THE ADDRESS DIRECTLY TO MEMORY.

* *Example shows parameters for a transfer to/from location 6000.*

REV. 16 FILE SYSTEM CHANGES

- 63 FILE UNITS PER USER (UNIT 53 RESERVED FOR COMOUTPUT)
- NEW CONFIG PARAMETER

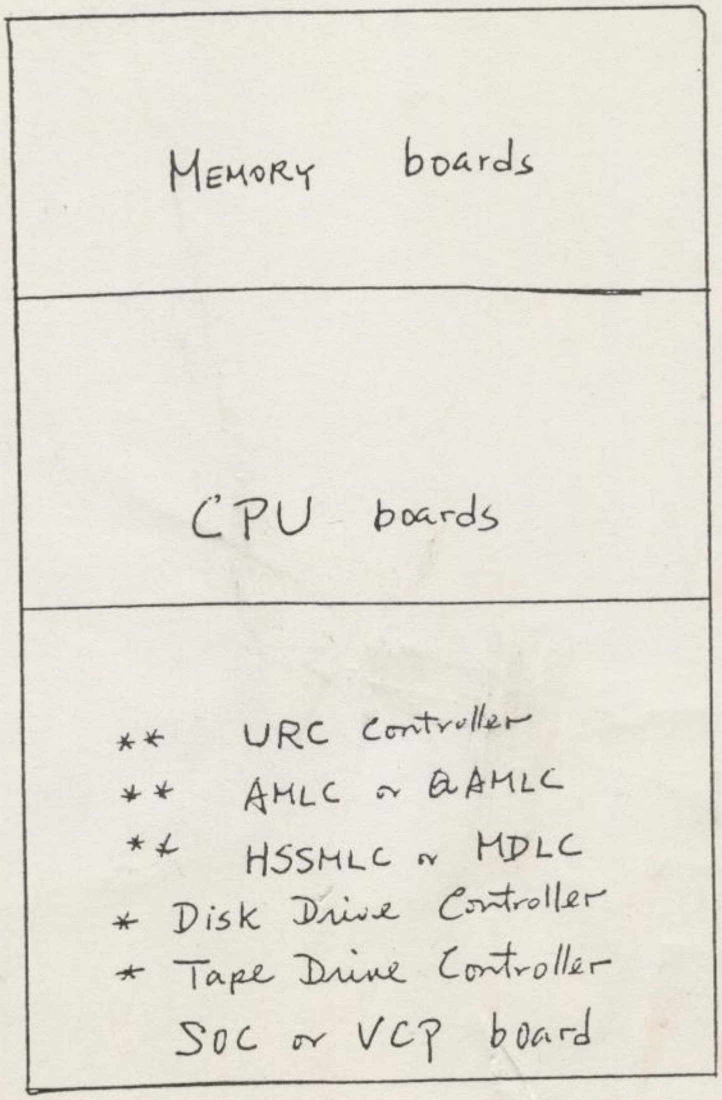
FILUNT	(RSVUNT)	(MAXUNT)	(TOTUNT
	(16)	(64)	(2048)

RSVUNT - NUMBER OF FILE UNITS GUARANTEED TO BE AVAILABLE TO EACH USER.

MAXUNT - MAXIMUM NUMBER OF UNITS A USER CAN HAVE OPEN.

TOTUNT - TOTAL NUMBER OF UNITS THAT MAY BE OPEN SIMULTANEOUSLY BY ALL USERS.

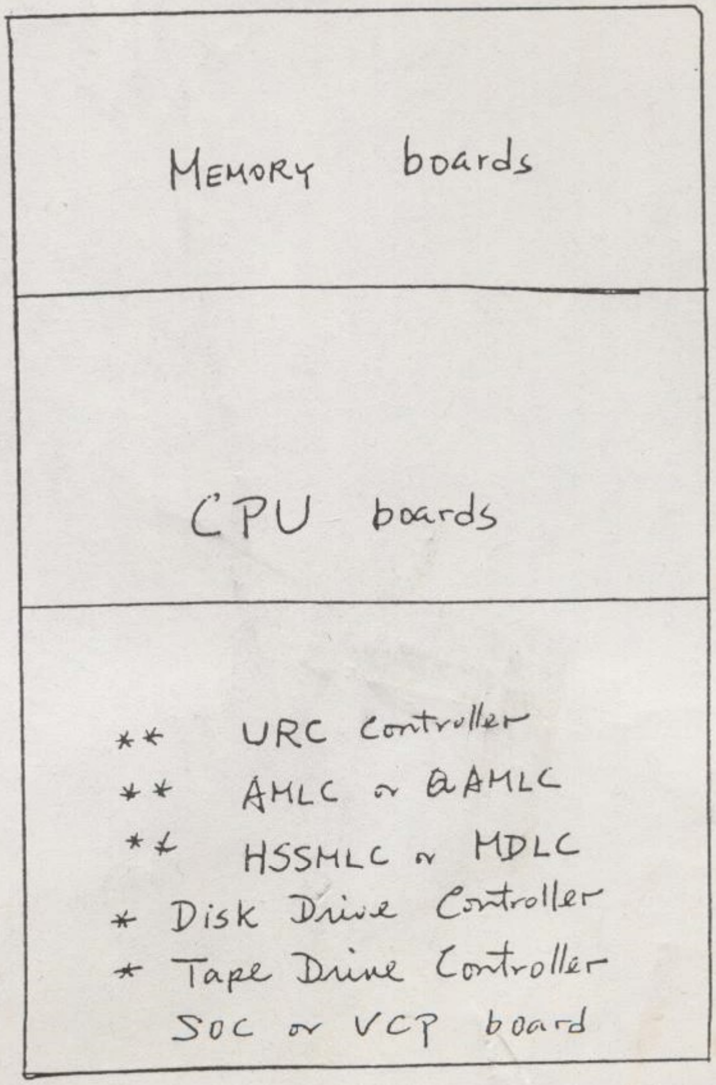
The Chassis



* Both boards positions are interchangeable

** These 3 boards' positions are interchangeable

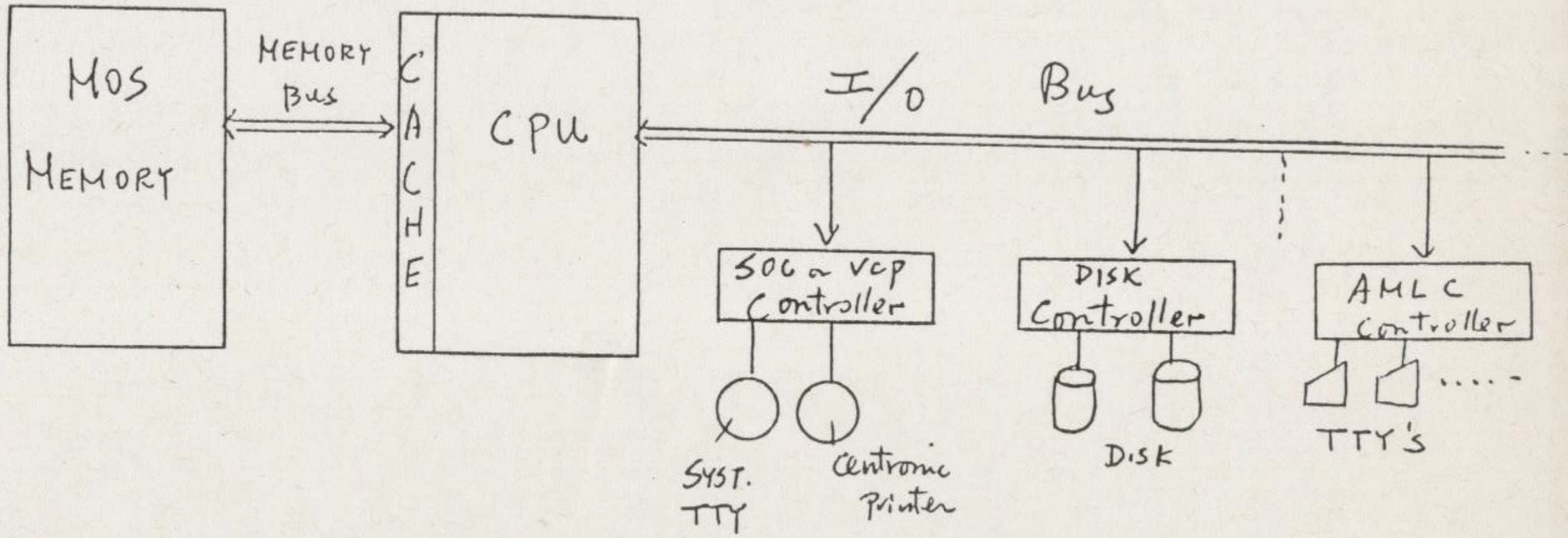
The Chassis



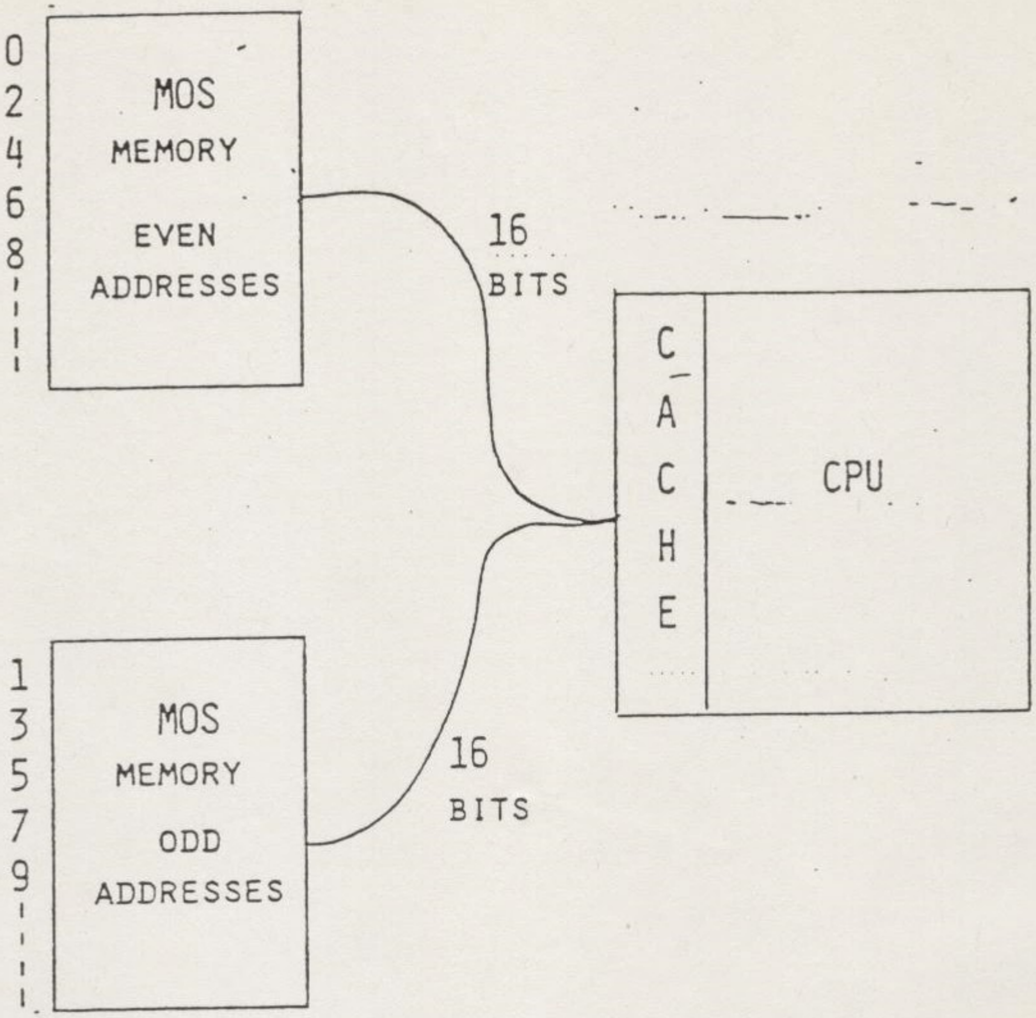
* Both boards positions are interchangeable

** These 3 boards' Positions are interchangeable

Machine Overview



INTERLEAVING



INTERLEAVING IS IMPLEMENT USING TWO IDENTICAL BOARDS.

ONE BOARD CONTAINS THE EVEN ADDRESSES, THE OTHER BOARD CONTAINS THE ODD ADDRESSES.

THIS HAS THE EFFECT OF SPEEDING UP SEQUENTIAL ACCESS AND REDUCES THE CACHE MISS RATE.

PRIME CENTRAL PROCESSOR SYSTEM CHARACTERISTICS

FEATURE	100	200	300	350	400	450	500	550	650	750
HIGHEST ADDRESS MODE SUPPORTED	64R	64R	64R	64V	64V	32I	32I	32I	32I	32I
MAXIMUM AMOUNT OF PHYSICAL MEMORY (BYTES)	128K	128K	512K	512K	8M	1M	8M	2M	4M	8M
MEMORY WORD SIZE (BITS)	16	16	16	16	16	16	16	16	16	32
MAXIMUM AMOUNT OF VIRTUAL MEMORY (BYTES)	n/a	n/a		2M	256M					
MAXIMUM AMOUNT OF VIRTUAL MEMORY/USER (BYTES)	n/a	n/a	128K	768K	512K	32M	512K	32M	32M	32M
MAXIMUM NUMBER OF USERS (INCLUDING SYSTEM)	1	1	32	32	64	64	64	64	64	64
CPU INTERNAL BUS AND REGISTER SIZE (BITS)	16	16	16	32	32	32	32	32	32	32
NUMBER OF REGISTERS IN REGISTER FILE	32	32	32	72	128	128	128	128	128	128
NUMBER OF (32 BIT) GENERAL PURPOSE REGISTERS	n/a	n/a	n/a	n/a	n/a	8	8	8	8	8
CACHE MEMORY SIZE (BITS x BYTES)	n/a	n/a	n/a	16x2K	16x2K	16x2K	16x2K	16x2K	16x2K	32x16
MULTI-RING MEMORY PROTECTION (PROGRAM SHARING)	N	N	N	Y	Y	Y	Y	Y	Y	Y
DMT and DMC	O	O	Y	Y	Y	Y	Y	Y	Y	Y
NUMBER OF DMA CHANNELS	8	8	8	8	32	32	32	32	32	32
DMQ SUPPORT	N	N	N	Y	Y	Y	Y	Y	Y	Y
PROCESS EXCHANGE	N	N	N	Y	Y	Y	Y	Y	Y	Y
INSTRUCTION PRE-FETCH	N	N	N	N	N	N	N	N	N	Y
MEMORY (BYTE) PARITY	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
PROCESSOR (BYTE) PARITY	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
ERROR CHECK AND CORRECTION (ECC) MEMORY	N	N	N	O	O	Y	Y	Y	Y	Y
INTER LEAVABLE MEMORY	N	N	N	Y	Y	Y	Y	Y	Y	Y
VIRTUAL CONTROL PANEL	N	N	N	Y		Y	Y	Y	Y	Y
BURST MODE I/O	N	N	N	N	N	N	N	N	N	Y
MICROVERIFY	N	O	Y	Y	Y	Y	Y	Y	Y	Y

EATURE	[illegible]	200	300	350	400	450	500	550	0	750
NUMBER OF INSTRUCTIONS IN INSTRUCTION SET	112	117	145	319	318	520	517	520	520	520
BUSINESS INSTRUCTION SET SUPPORT	N	N	N	U	U	H	H	H	H	H
HARDWARE MULTIPLY/DIVIDE and DOUBLE PRECISION ARITHMETIC	O	O	Y	Y	Y	Y	Y	Y	Y	Y
SINGLE and DOUBLE PRECISION FLOATING POINT ARITHMETIC	N	O	O	Y	Y	Y	Y	Y	Y	Y
32 BIT ARITHMETIC LOGIC UNIT	N	N	N	Y	Y	Y	Y	Y	Y	Y
32 BIT INTEGER ARITHMETIC	N	N	N	Y	Y	Y	Y	Y	Y	Y
64 BIT INTEGER ARITHMETIC	N	N	N	Y	Y	Y	Y	Y	Y	Y
FAST FLOATING POINT ARITHMETIC (MICROCODE)	N	N	N	N	N	Y	Y	Y	Y	Y
NUMBER OF BOARDS IN CENTRAL PROCESSOR	1	1	1	1	2	2	3	2	3	5

O	- OPTIONAL
N	- NOT SUPPLIED OR SUPPORTED
Y	- YES IT IS SUPPLIED OR SUPPORTED
U	- SUPPORTED BY UNIMPLEMENTED INSTRUCTION SOFTWARE PACKAGE
H	- SUPPORTED BY HARDWARE (OR FIRMWARE)
n/a	- THIS FEATURE DOES NOT APPLY TO THIS CPU

Register Files

THE CPU INCORPORATES A HIGH SPEED REGISTER FILE OF 128 LOCATIONS, EACH 32 BITS.

THESE LOCATIONS ARE DIVIDED INTO 4 GROUPS AS FOLLOWS:

GROUP 0 (FILE ADDRESSES 0-'37)
USED BY MICROCODE AND SYSTEM

GROUP I (FILE ADDRESSES '40-'77)
32 DMA CHANNEL REGISTERS

GROUP II (FILE ADDRESSES '100-'137)
USER REGISTER SET A

GROUP III (FILE ADDRESSES '140-'177)
USER REGISTER SET B

TWO USER REGISTER SETS ARE INCLUDED TO FACILITATE FAST PROCESS EXCHANGE. ONE SET IS AVAILABLE TO THE CURRENTLY RUNNING PROCESS AND IS REFERRED TO AS THE CURRENT REGISTER SET.

DETAILS OF THE USER REGISTER SET ARE AS FOLLOWS:

RFØ SCRATCH

ABS	HI	LO
0	TRØ	
1	TR1	
2	TR2	
3	TR3	
4	TR4	
5	TR5	
6	TR6	
7	TR7	
10	RDMX1	
11	RDMX2	
12		RATMPL
13	RSGT1	
14	RSGT2	
15	RECC1	
16	RECC2	
17		REOIV
20	ZERO	ONE
21	PBSAVE	
22	RDMX3	
23	RDMX4	
24	C377	
25		
26		
27		
30	PSWPB	
31	PSWKEYS	
32	PPA: PLA	PCBA
33	PPB: PLB	PCBB
34	DSWRMA	
35	DSWSTAT	
36	DSWPB	
37	RJAYPTR	

RF1 DMA

ABS	HI	LO
40		
41		
42		
43		
44		
45		
46		
47		
50		
51		
52		
53		
54		
55		
56		
57		
60	(20)	(21)
61		
62	(22)	(23)
63		
64	(24)	(25)
65		
66	(26)	(27)
67		
70	(30)	(31)
71		
72	(32)	(33)
73		
74	(34)	(35)
75		
76	(36)	(37)
77		

RF2 CURRENT REG SET 2

CRS	ABS	HI	LO
0	100	GRØ:OLT2	
1	101	GR1:PTS	
2	102	GR2(1,A,LH)	(2,B,LL)
3	103	GR3 (EH)	(EL)
4	104	GR4	
5	105	GR5(3,S,Y)	
6	106	GR6	
7	107	GR7(Ø,X)	
10	110	FAR1 (13)	
11	111	FRL1	
12	112	FAR2 (4)	(5)
13	113	FRL2:VSC(6)	
14	114	PB	
15	115	SB (14)	(15)
16	116	LB (16)	(17)
17	117	XB	
20	120	DTAR3 (10)	
21	121	DTAR2	
22	122	DTAR1	
23	123	DTARØ	
24	124	KEYS	(MODALS)
25	125	OWNER	
26	126	FCODE (11)	
27	127	FADDER	(12)
30	130	TIMER	
31	131		
32	132		
33	133		
34	134		
35	135		
36	136		
37	137		

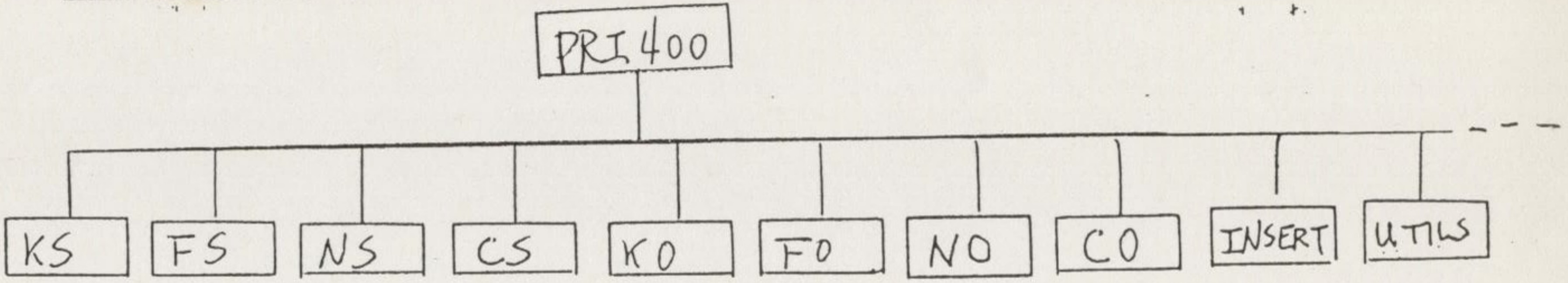
RF3 CURRENT REG SET 3

CRS	ABS	HI	LO
0	140	GRØ:OLT2	
1	141	GR1: PTS	
2	142	GR2(1,A,LH)	(2,B,LL)
3	143	GR3 (EH)	(EL)
4	144	GR4	
5	145	GR5(3,S,Y)	
6	146	GR6	
7	147	GR7(Ø,X)	
10	150	FAR1(13)	
11	151	FRL1	
12	152	FAR2(4)	(5)
13	153	FRL2:VSC(6)	
14	154	PB	
15	155	SB (14)	(15)
16	156	LB(16)	(17)
17	157	XB	
20	160	DTAR3 (10)	
21	161	DTAR2	
22	162	DTAR1	
23	163	DTARØ	
24	164	KEYS	(MODALS)
25	165	OWNER	
26	166	FCODE(11)	
27	167	FADDER	(12)
30	170	TIMER	
31	171		
32	172		
33	173		
34	174		
35	175		
36	176		
37	177		

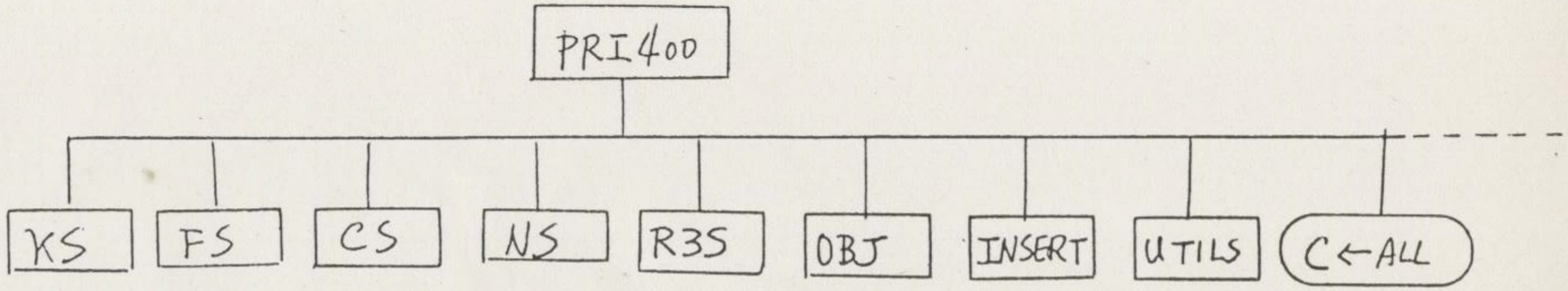
P-400/500 REGISTER FILES

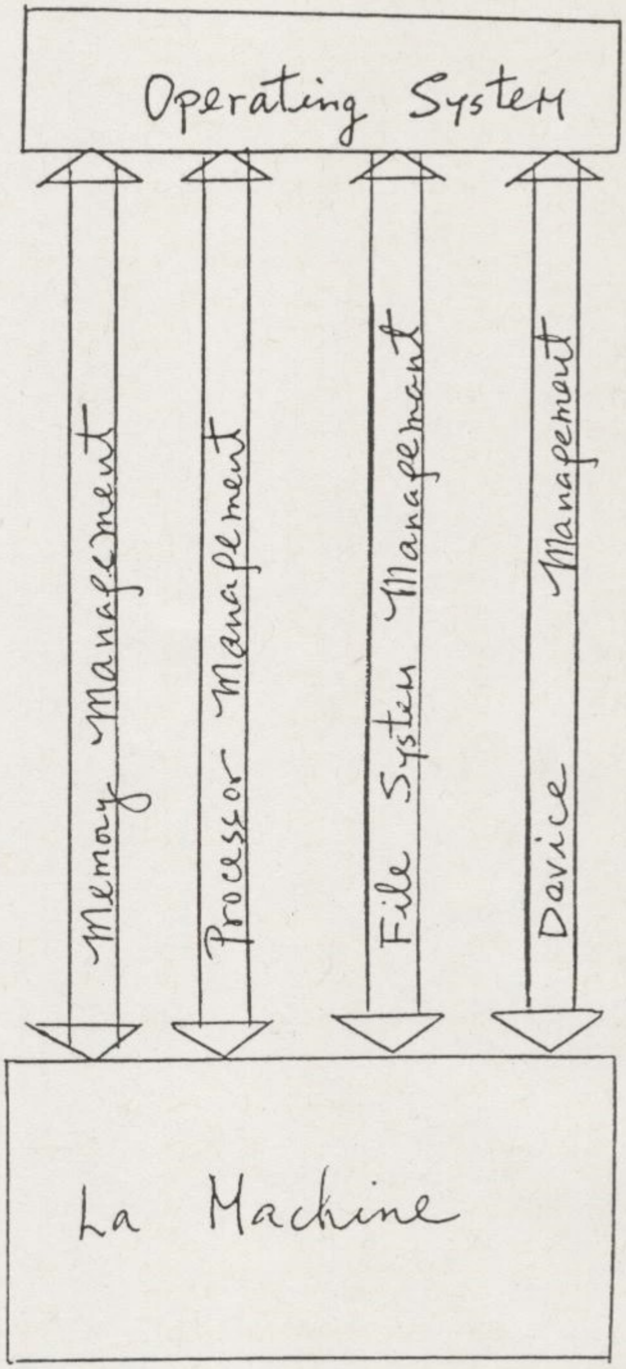
PRIMOS STRUCTURE

Rev. 16


PRI400
KS
FS
NS
CS
KO
FO
NO
CO
INSERT
UTILS

Rev. 17


PRI400
KS
FS
CS
NS
R3S
OBJ
INSERT
UTILS
C←ALL


Operating System
Memory Management
Processor Management
File System Management
Device Management
La Machine

REV. 16 PRIMOS IV SEGMENT LAYOUT

SEG Ø I/O SEGMENT

PAGE NUMBER (OCTAL)	
'0	DMC CHANNELS FOR AMLC, SMLC, MAG. TAPE. AMLC DEDICATED CELLS AND TUMBLE TABLES, SMLC STATUS BUFFERS
	'1370 DVDISK
'2	QAMLC Q CONTROL BLOCKS
'3	PRBUFØ, CRBUFF, CPBUFF
'4	PRBUF1, CR2BUFF, CP2BUFF
'5	PRBUF2, PRBUF3
'6	VGBUFF
'10	UNUSED
'35	RING NODE WINDOW
'41	SECOND MAG. TAPE CONTROLLER WINDOW
'47	SMLC WINDOW
'63	IPC WINDOW
'65	MAG. TAPE DUMP WINDOW
'66	FIRST MAG. TAPE CONTROLLER WINDWO
'74	DISK WINDOW
'77	

SEG 1 FILE SYSTEM ASSOCIATIVE BUFFERS

SEG 2 & 3 MOVU2U SEGMENT WINDOWS

SEG 4 INTERRUPT SEGMENT

- o PHANTOM INTERRUPT CODE
- o CHECK HEADERS
- o SEMCOM - SYSTEM SEMAPHORES 0 - '2000
- o READY LIST
- o WARM START CODE
- o COLD START CODE
- o ECC HANDLER
- o (OPERATING SYSTEM VPSD) '2000 - '12000
- o INTERRUPT FAULT TABLE AND HANDLERS
- o COMMON CHECK HANDLER '76000 -
- o FIRST LEVEL EVENT LOGGER (LOGEVI) '116000
- o PCB's
- o CONCEALED STACKS
- o INTERRUPT STACK

SEG 5 RING Ø GATE SEGMENT

SEG 6

- TMAIN
  - SUPERVISOR COMMON (SUPCOM)
  - CLOCK PROCESS
  - USER FAULT TABLE AND HANDLERS
  - SVC INTERLUDES AND CODE
  - COMXIT, UNLOAD, ETC
- KERNEL PROCEDURES
  - DEVICE DRIVERS
  - LOCK MECHANISM
  - BUFFER CONTROL (TFLIOB, LOCATE, ETC)
  - PAGE TURNER
  - COLD START CODE (AINIT, AMINIT)
  - COMMAND PROCESSOR (DOSSUB)
  - BACKSTOP PROCESS (SCHED)

SEG 7

- TERMINAL I/O BUFFERS
- SPECIAL BUFFERS (PTR, PTP, CEN)
- Q DATA BLOCKS FOR QAMLC

SEG 10

- PER USER DATA (USRCOM)
- FILE SYSTEM UNIT TABLES (UTCOM)

SEG 11	FILE SYSTEM PROCEDURES
SEG 12	NETWORK DATA AND PROCEDURES SMLC DATA AND PROCEDURES
(SEG 13	COMMAND ENVIRONMENT)
SEG 14	(ONE TO ONE) o RSAV AREA o OPERATING SYSTEM VPSD ENTRY o CONFIGURATION COMMON (FIGCOM AT '700) o MAGTAPE DUMP AND MEMORY SCAN o WARM AND COLD START ENTRIES o VIRTUAL MEMORY MECHANISM CODE o MEMORY MAP (MMAP) o PAGE MAPS (HMAP) o PTUSEG o SEGMENT DESCRIPTOR TABLES
SEG 6000	SUPERVISOR RING Ø STACK

REV. 16 PRIMOS IV USEFUL LOCATIONS

MEMORY MAP	MMAP	14/2000
START OF PAGEMAPS	HMAP	14/12000
START OF PTUSEG	PTUSEG	14/140000
NO. SEGMENTS IN SYSTEM	NSEG	14/141200
NO. SEGMENTS PER USER	NUSEG	14/141201
NO. OF CONFIGURED USERS	NUSR	6/2207
PAGE FAULT COUNTER (32 BITS)	PFCN	6/2334
LOCATE READ COUNTER (32 BITS)	LOCCNT	6/2336
SCHEDULING CONSTANT	MAXSCH	6/2213
ELIGIBILITY TIME SLICE	ELIGTS	6/2321
TIMESLICE FOR USER N	USRTS	6/1277 + N
HIGH PRIORITY Q	HIPRI	4/536
ELIGIBILITY Q	ELIGQ	4/540
LOW PRIORITY Q FOR DEFAULT - USER LEVEL	LOWPRI	4/550
DEFAULT USER LEVEL ON READY LIST	LEVEL	4/626
PCB's :	CLOCK	4/75700

REV. 16 PRIMOS IV USEFUL LOCATIONS (CONTINUED)

PCB's:	AMLC	4/77100
	BACKSTOP	4/76600
	USER 1	4/100100
	USER N	4/100000 + 100N
LOCKS:	FSLOK	6/13543
	UFDLOK	6/13551
	UTLOK	6/13557
	TRNLOK	6/13565
	RATLOK	6/13573
	PAGLCK	6/13651
	PAGSEM	4/532
	DSKLCK	6/13667
	DSKSEM	4/534
FIGCOM		14/700
BMQMSK		14/723

REV. 16 PRIMOS IV SEMAPHORE LOCATIONS

HIPRI Q	4/536
ELIG Q	4/540
LOWPRI Q	4/542 - 552
INPUT WAIT (BUFSEM)	5/17524 - 17724
TIMED WAIT (CLKRNG)	6/2350 - 2374
FILE SYSTEM	6/13543 - 13576
PAGE IN TRANSITION (PAGSEM)	4/532
DVDISK WAIT (DSKLCK)	6/13667 - 13672
DISK I/O (DSKSEM)	4/534
LOCATE WAIT (LOCSEM)	6/13675
USER SEMAPHORES	5/21045 - 21245
NETWORK WAIT	6/20136 - 20336
MAG TAPE WAIT	6/21247 - 21261

SYSTEM LIMITS EXPANDED (Rev. 17)

. 64 SHARED SEGMENTS:

2000	ED (VII IN SEG 13)
2001 - 2003	DBMS
2004 - 2011	SPSS
2012	DBMS
2013	BASICV
2014	KIDA, FORMS, COBOL, SHARED LIBRARIES
2015	DPTX - TCF
2020	MIDAS SEMAPHORES
2030 - 2037	RESERVED FOR USER APPLICATIONS
2040 - 2042	DBG
2050	FTN SHARED LIBRARY

. 511 TOTAL SEGMENTS (NSEG). DEFAULT IS STILL 192
64 MB.

. 128 FUNITS (IOCS STILL 16)

. 62 STARTED UP DISKS

PRIMOS MEMORY REQUIREMENTS

. WIRED:

SEGNO	REV 17
0	3 K WORDS
4	4
6	16
14	4
22	3
600D	1

. PLUS

- . SEG 4 - 100 WORDS FOR EACH CONFIGURED USER
  (PCB'S AND CONCEALED STACKS)
- . SEG 7 - TERMINAL I/O BUFFERS FOR EACH CONFIGURED USER
  (DEFAULT 96 AND 192 WORDS RESPECTIVELY)
  - PAPER TAPE, CENTRONICS BUFFERS AS REQUESTED
  (1K WORDS)
- . SEG 12- 6K WORDS FOR MDLC
  18K WORDS FOR PNC
  23K WORDS FOR MDLC & PNC
- . SEG 14-SEGMENT DESCRIPTOR TABLES (NUSEG*2* NUMBER
  CONFIG. USERS)
  -MMAP, 1K WORDS FOR EACH 2MB OF PHYS. MEMORY
- ' SEG 21- Q DATA BLOCKS FOR EACH CONFIG. LINE IF
  QAMLC PRESENT (DEFAULT 32 WORDS/LINE)
- . SEG 22- PAGE MAPS, 128 WORDS FOR EACH SEGMENT IN
  USE ABOVE '1777
- . SEG 6000 - RING Ø STACK, 1K WORDS FOR EACH LOGGED IN
  USER.

PRIMOS MEMORY REQUIREMENTS (CONT.)

- 3K WORDS MORE THAN REV 16 ([illegible])

- EXAMPLES:

  10 USERS CONFIG., 5 LOGGED IN - 48K WORDS WIRED
  20 USERS CONFIG., 10 LOGGED IN- 61K WORDS WIRED
  30 USERS CONFIG., 15 LOGGED IN- 73K WORDS WIRED

- WIRMEM CONFIG. DIRECTIVE PRINTS INITIAL WIRED MEMORY, NEED TO ADD USERS RING Ø STACKS AS THEY LOGIN, PAGE MAPS AS THEY ARE USED, BUFFERS AS DEVICES ARE USED.

PRIMOS MEMORY REQUIREMENTS (CONT.)

. PAGED:

	SEGNO	REV 17	REV 16
ASSOCIATED BUFFERS	1	64	64 K WORDS
ECB'S	5	2	2
KERNEL CODE	6	36	26
USRCOM, UTCOM	10	82 WORD PER CONFIG. USER +16 WORDS PER FILE UNIT IN USE.	
FILE SYSTEM CODE	11	19	19
NETWORK, COMMS. CODE	12	38	37
COMMAND ENVIRONMENT CODE	13	34	0
DPTX	15	44	0
RING 3 STACK	6002	1+ PER CONFIG USER	0

. WORKING SET:

- . MAIN CHANGE OVER REV 16 IS THE NEW COMMAND ENVIRONMENT
  - ADDITIONAL 10K WORDS FOR SEG 13
  - PLUS 1 1/2K WORDS PER "ACTIVE" USER
- . GUIDELINE: 20→30K WORDS INCREASE

. REV 17 SHOULD NOT BE RUN ON SYSTEMS WITH LESS THAN 1/2MBYTE PHYSICAL MEMORY.

PRIMOS SEGMENT LAYOUT (REV 17.1)

SEG 0 . I/O SEGMENT

- . DMC CHANNELS FOR AMLC, SMLC, MAG TAPE
- . AMLC DEDICATED CELLS AND TUMBLE TABLES
- . SMLC STATUS BUFFERS
- . DISK CHANNEL PROGRAMS
- . Q CONTROL BLOCKS FOR QAMLC

SEG 1 . FILE SYSTEM ASSOCIATIVE BUFFERS

SEG 2 & 3 . MOVU2U SEGMENT WINDOWS

SEG 4 . INTERRUPT SEGMENT

- . PHANTOM INTERRUPT CODE
- . CHECK HEADERS
- . SEMCOM - SYSTEM SEMAPHORES
- . READY LIST 0-'1777
- . WARM START CODE
- . COLD START CODE
- . ECC HANDLER
- . (OPERATING SYSTEM VPSD) '2000 - '13777
- . COMMON CHECK HANDLER
- . FIRST LEVEL EVENT LOGGER
- . PCB'S '76000 -
- . CONCEALED STACKS '115777
- . INTERRUPT FAULT TABLE AND HANDLERS
- . INTERRUPT STACK

SEG 5 . RING Ø GATE SEGMENT

PRIMOS SEGMENT LAYOUT (REV 17.1) (CONT.)

SEG 6 . TMAIN

- . SUPERVISOR COMMON (SUPCOM)
- . CLOCK PROCESS
- . RING Ø FAULT TABLE AND HANDLERS
- . UNLOAD, SEM$, MOV . . .ETC.

. KERNEL PROCEDURES

- . DEVICE DRIVERS (INCLUDING DISKIO)
- . LOCK MECHANISM
- . BUFFER CONTROL (TFLIO$, LOCATE ETC.)
- . PAGE TURNER
- . COLD START CODE (AINIT, AMINIT)
- . DOSSWB
- . INTERNAL LOGIN
- . BACKSTOP PROCESS(SCHED)

SEG 7 . TERMINAL I/O BUFFERS

. SPECIAL BUFFERS (PTR, PTP, CEN)

SEG 10 . PER USER DATA (USRCOM)

. FILE SYSTEM UNIT TABLES (UTCOM)

SEG 11 . FILE SYSTEM PROCEDURES

SEG 12 . NETWORK DATA AND PROCEDURES

. MDLC DATA AND PROCEDURES

SEG 13 . COMMAND ENVIRONMENT CODE

. CONDITION MECHANISM CODE

. RING 3 FAULT TABLE AND HANDLERS

. SVC INTERLUDES AND CODE

PRIMOS SEGMENT LAYOUT (REV 17.1) (CONT.)

SEG 14	. RSAV AREA ('200)
	. OPERATING SYSTEM VPSD ENTRY
	. CONFIGURATION COMMON (FIGCOM AT '700)
	. MAG TAPE DUMP AND MEMORY SCAN
	. WARM AND COLD START ENTRIES
	. VIRTUAL MEMORY MECHANISM CODE
	. MEMORY MAP (MMAP) ('2000)
	. PTUSEG ('150000)
	. SEGMENT DESCRIPTOR TABLES
SEG 15-20	. DPTX
SEG 21	. Q DATA BLOCKS FOR QAMLC'S
SEG 22	. PAGE MAPS
SEG 6000	. RING Ø STACK
SEG 6001	. SHARED LIBRARY IMPURE SECTIONS
	. ABBREVIATION FILE
SEG 6002	. RING 3 STACK

SYSTEM LIMIT EXTENSIONS (Rev 18)

. RING BUFFERS MAY BE UP TO TWO SEGMENTS LONG. USE BOTH SEGMENT '7 AND SEGMENT '34.

. NSEG LIMIT NOW 1022 SEGMENTS

. NUMBER OF SHARED [DTAR 1] SEGMENTS INCREASED FROM 64 TO 128. ['2000-'2177]

. NUMBER OF SHARED LIBRARIES INCREASED TO 16

. PAGE DISK SIZE INCREASED FROM 512 SEGMENTS TO ENTIRE 300MB, IF NEEDED.

VIRTUAL MEMORY DATA STRUCTURE CHANGES

. AT REV 17 HMAP/LMAP COULD SUPPORT 511 ('777) SEGMENTS.

. BY PUTTING HARDWARE MAPS IN SEGMENT 22 AND LOGICAL MAPS IN SEGMENT 33 WE CAN NOW SUPPORT 1022 SEGMENTS ('776). START AT WORD '100.

. PTUSEG LARGER. NOW STARTS AT 14/25200.

. MMAP INCREASED TO 2 WORDS/ENTRY; STARTS AT 14/4000

EXTRA WORD USED BY

* * * *

* * *VMFA * * *

* * * *

VMFA

. METHOD OF PAGING DIRECTLY FROM FILE SYSTEM

. AT REV 18 ONLY ENOUGH SUPPORT FOR POSSIBLE EARLY RELEASE OF EPF's.

. TWO NEW KEYS TO SRCH$$

:20 OPEN DAM FILE FOR VMFA READ ACCESS

:60 OPEN DAM FILE FOR VMFA WRITE ACCESS

TO USE AT REV 18

1. CALL SRCH$$ TO OPEN IN VMFA MODE.
2. CALL VINIT$ TO MAP FILE TO MEMORY.
3. CALL SRCH$$ TO FREE UNIT.
4. PROCESS FILE.
5. CALL RTNSEG TO REMOVE SEGMENTS.

VINIT$-

CALL VINIT$ (KPY, UNIT, LOC (SEGTAB), LOC (RSEGTAB), NSEGS, LOC (WINDOW), LOC (ACCESS), 'OC (LEN), CODE)

KEY - :10 CONSECTIVE SEGNOS REQUIRED

:4 WILL ACCEPT ANY OLD SEGMENTS

:2 I AM RECOMMENDING SOME SEGMENTS

:1 I MUST HAVE SPECIFIC SEGMENTS

UNIT - UNIT ON WHICH FILE IS OPEN

SEGTAB - SEGMENT NUMBER(S) MAPPED (RETURNED)

RSEGTAB - RECOMMENDED SEGMENT NUMBER(S)

NSEGS - NUMBER OF SEGMENTS TO MAP

WINDOW - WINDOW NUMBER IN FILE (FIRST SEGMENT Ø, SECOND SEGMENT 1, ETC.)

ACCESS _ ACCESS RIGHTS DESIRED FOR EACH SEGMENT

LEN - LENGTH OF DATA IN EACH SEGMENT (RETURNED)

CODE - STANDARD ERROR CODE (ERRD.F UPDATED FOR VMFA)

- MUST USE NVMFS CONFIGURATION DIRECTIVE

  NVMFS MAY BE FROM 1-256

  NSEG + NVMFS MUST NOT BE GREATER THAN 1022

IF VMFA SEGMENT, PTUSEG ENTRY IS AFTER THE NSEG'TH ENTRY. WHEN NOT IN MEMORY, LMAP CONTAINS THE LOW ORDER RA OF PAGE - HMAP CONTAINS THE HIGH ORDER. WHEN PAGE IS IN MEMORY, HIGH ORDER RA IS STORED IN THE SECOND WORD OF THE MMAP ENTRY.

Rev. 18

SEG	0	I/O MAP SEGMENT
SEG	1	LOCATE BUFFER SEG
SEG	2-3	TEMP SEGS - INTERUSER MOVES
SEG	4	CHECKS, TRAPS, PX, ETC.
SEG	5	RING 0 GATES
SEG	6	RING 0 KERNEL CODE, LINKAGE
SEG	7	LOW SPEED I/O BUFFERS
SEG	10	FILE SYSTEM DATA STRUCS
SEG	11	FILE SYSTEM CODE, LINKAGE, OVERFLOW FROM SEG 6
SEG	12	NETWORK CODE, LINKAGE
SEG	13	COMMAND LOOP SEGMENT 1
SEG	14	COLD&WARM START, SDW0,1, ETC
SEG	15-20	USED BY DPTX
SEG	21	USED BY DMQ BUFFER
SEG	22	PAGE MAP SEGMENT
SEG	23-26	SMLC COPY SEGMENTS
SEG	27	NETWORK BUFFERS
SEG	30	NETWORK QUEUES/BHA'S
SEG	31	NETWORK
SEG	32	COMMAND LOOP SEGMENT 2
SEG	33	LOGICAL PAGE MAP SEGMENT
SEG	34	SECOND SEGMENT FOR RING BUFFERS
SEG	35-37	FREE
SEG	40-237	USERS WIRED RING0 STACKS
SEG	240-277	NETWORK MAPPED SEGMENTS
SEG	6000	WIRED RING0 STACK
SEG	6001	ABBREVS - DYNAMIC LINKS
SEG	6002	RING3 STACK
SEG	6003	UNWIRED RING0 STACK
SEG	6004	CPL
SEG	6005	GLOBAL VARIABLES

MEMORY MANAGEMENT

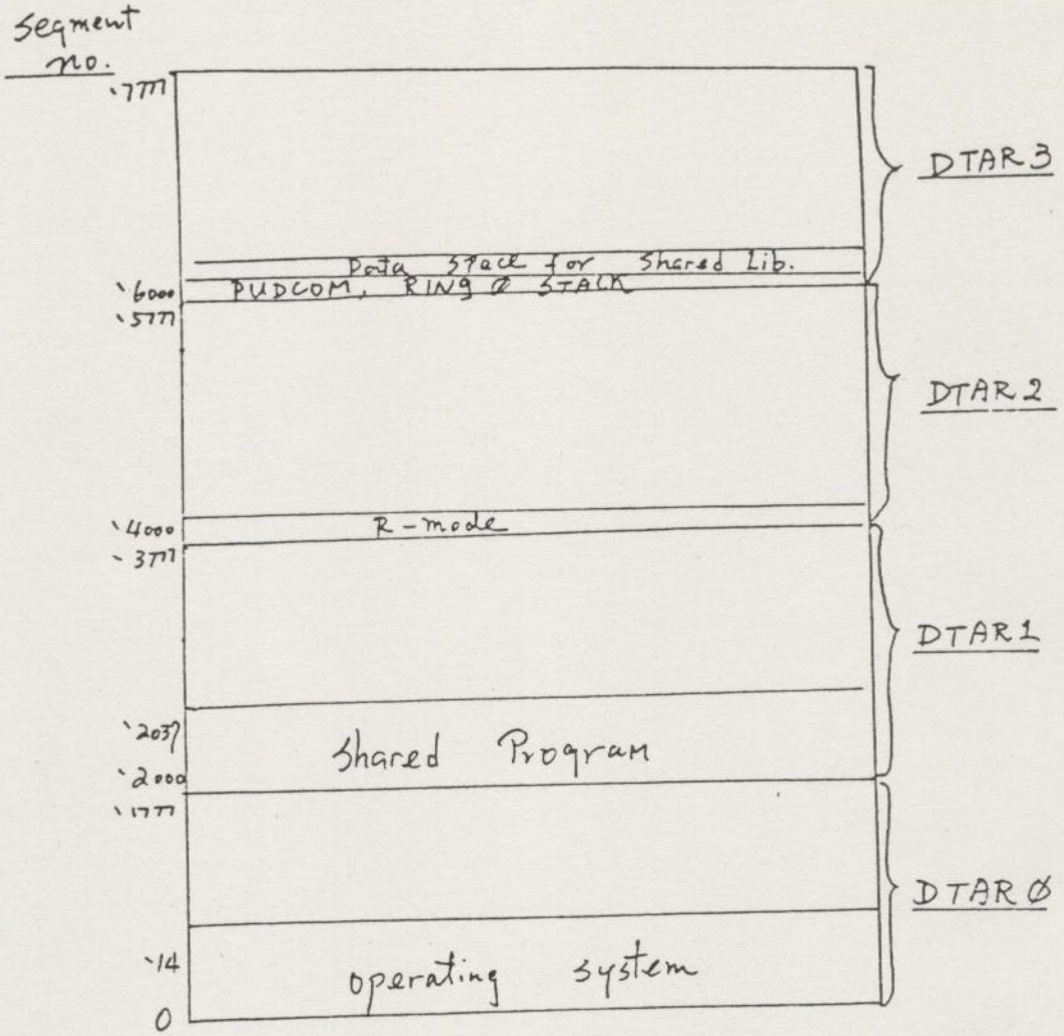
In this section, we shall cover:

- What is Virtual Memory?

- How the system manages its memory?

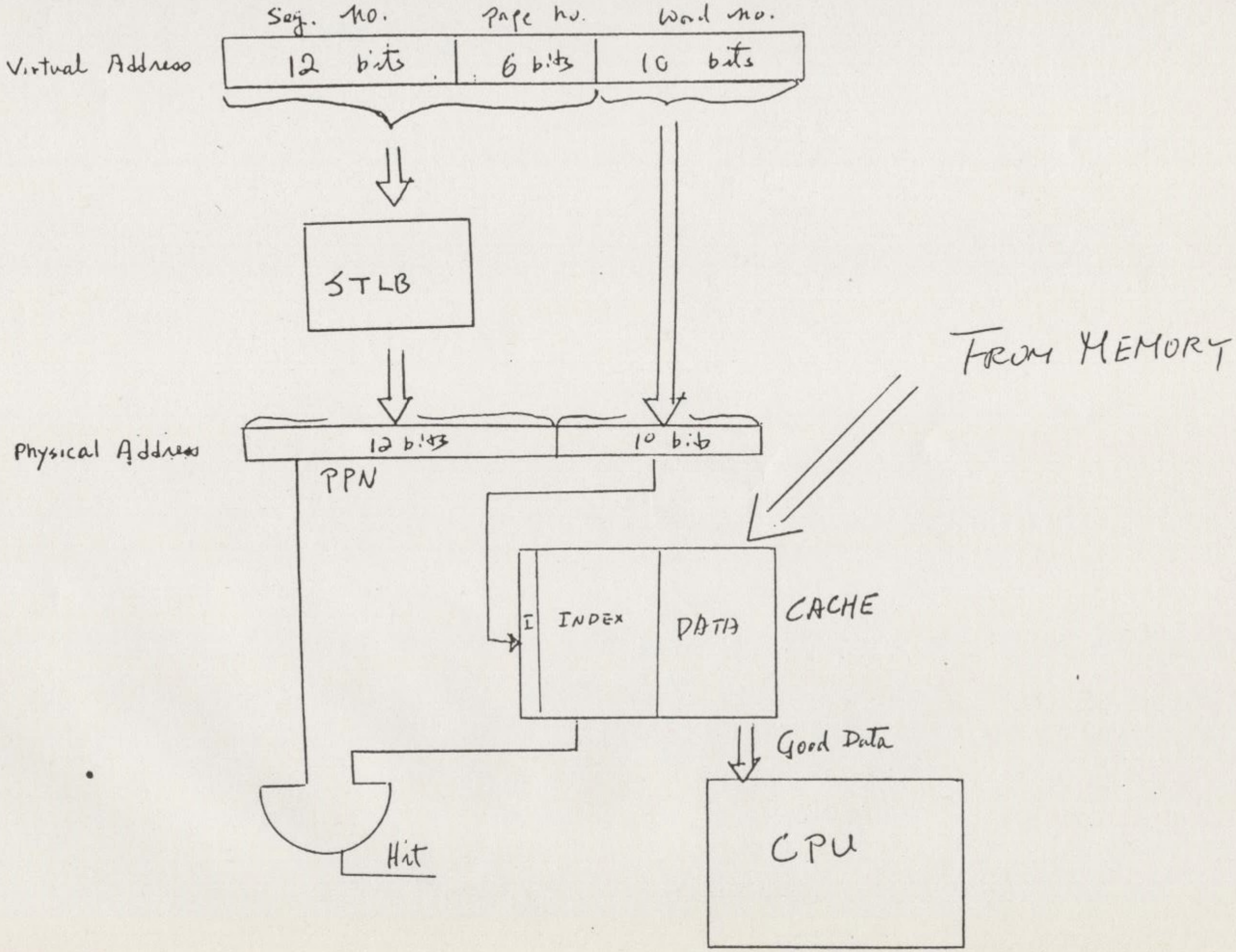
- How does a virtual address translate into a physical address?

Virtual Memory are divided into 4 groups of 1024 segments. Each group has 1 Descriptor Table Address Register (DTAR)

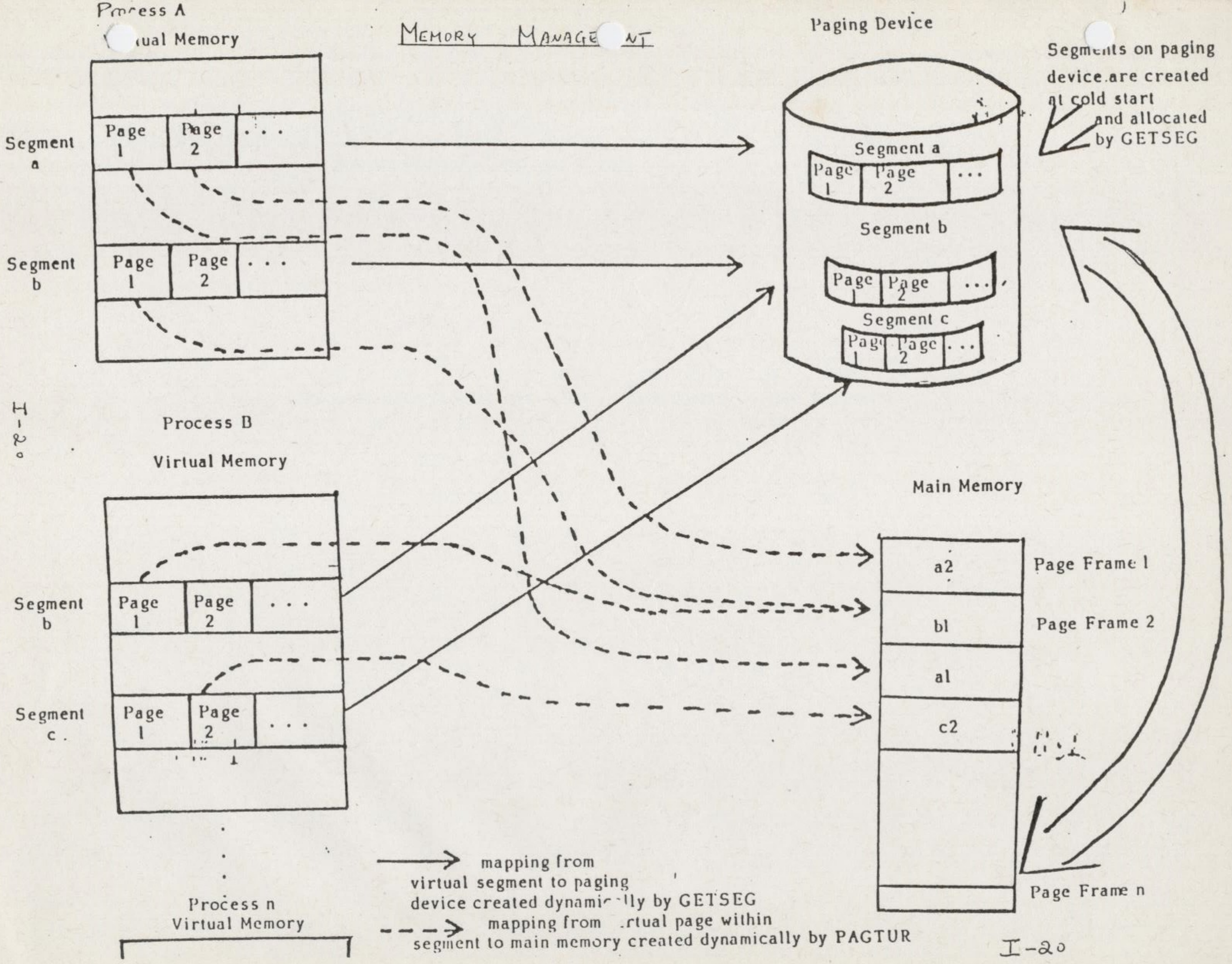


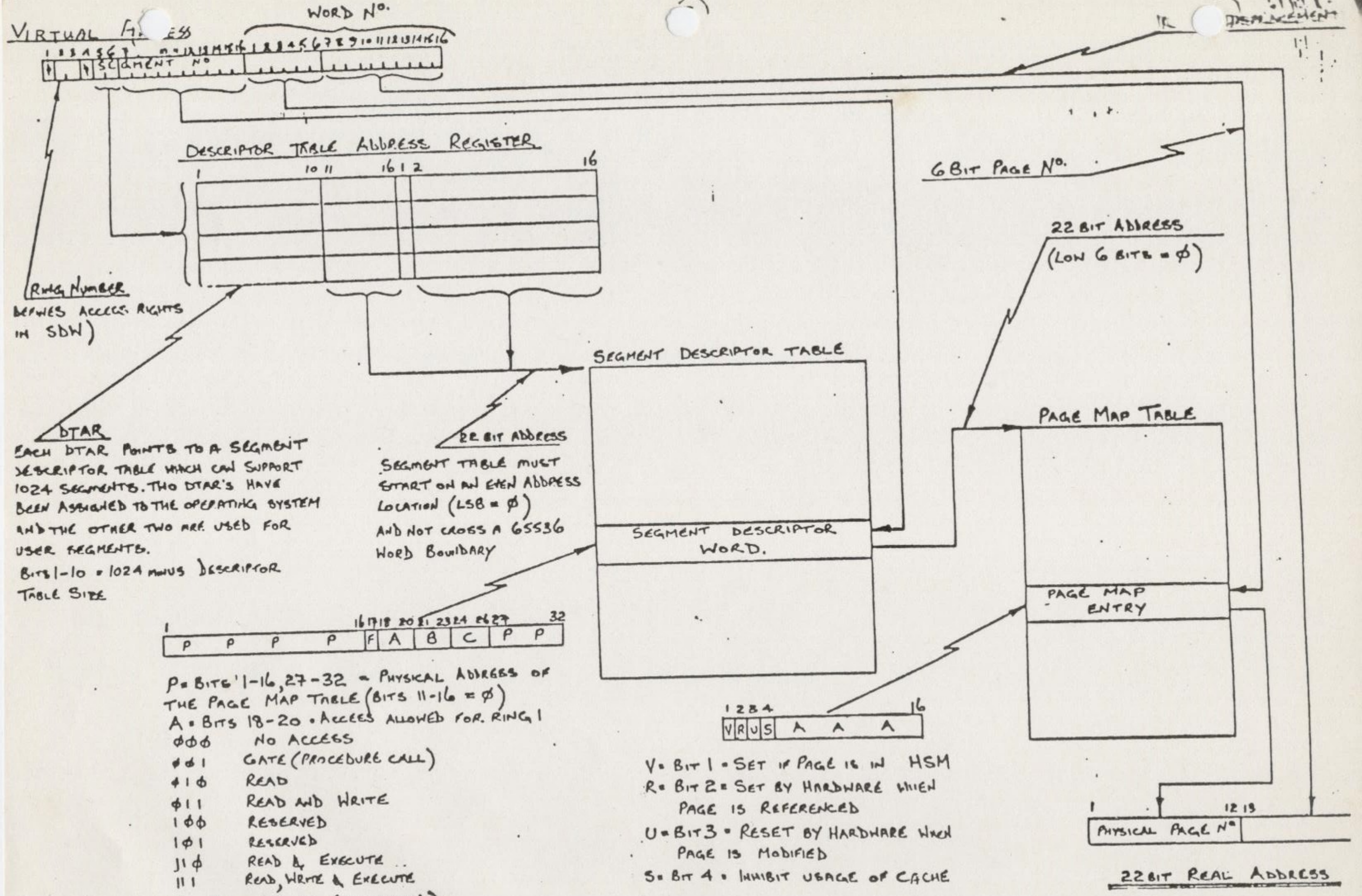
DTAR Ø — Used by operating system
DTAR 1 — shared by all users
DTAR 2 } Private to user.
DTAR 3 }

Address Translation



MEMORY MANAGEMENT





RING NUMBER DEFINES ACCESS RIGHTS IN SDW)

DTAR
EACH DTAR POINTS TO A SEGMENT DESCRIPTOR TABLE WHICH CAN SUPPORT 1024 SEGMENTS. TWO DTAR'S HAVE BEEN ASSIGNED TO THE OPERATING SYSTEM AND THE OTHER TWO ARE USED FOR USER SEGMENTS.

BITS 1-10 = 1024 MINUS DESCRIPTOR TABLE SIZE

SEGMENT TABLE MUST START ON AN EVEN ADDRESS LOCATION (LSB = ϕ) AND NOT CROSS A 65536 WORD BOUNDARY

P = BITS 1-16, 27-32 = PHYSICAL ADDRESS OF THE PAGE MAP TABLE (BITS 11-16 = ϕ)

A = BITS 18-20 = ACCESS ALLOWED FOR RING 1

ϕϕϕ	NO ACCESS
ϕϕ1	GATE (PROCEDURE CALL)
ϕ1ϕ	READ
ϕ11	READ AND WRITE
1ϕϕ	RESERVED
1ϕ1	RESERVED
11ϕ	READ & EXECUTE
111	READ, WRITE & EXECUTE

B = BITS 21-23 (RESERVED)

C = BITS 24-26 = ACCESS ALLOWED FOR RING 3 (SAME CODE AS ABOVE)

V = BIT 1 = SET IF PAGE IS IN HSM

R = BIT 2 = SET BY HARDWARE WHEN PAGE IS REFERENCED

U = BIT 3 = RESET BY HARDWARE WHEN PAGE IS MODIFIED

S = BIT 4 = INHIBIT USAGE OF CACHE

NOTE: RING ϕ ALWAYS HAS CODE 111

A PAGE MAP TABLE HAS 64(100) ENTRIES

PROCEDURE CALL

PROCEDURE/LINKAGE/STACK ARCHITECTURE:

MOTIVATION IS SHARED CODE

NEED SEPARATION OF CODE AND DATA

DEFINE THREE MEMORY CLASSES FOR EACH PROCEDURE :

(1) PROCEDURE AREA:
- .1 PER SYSTEM
- . PURE CODE
- . LITERALS
- . READ ONLY AREA
- . POINTED TO BY PB

(11) LINKAGE AREA:
- .1 PER USER
- . FORTRAN LOCAL VARIABLES
- . LINKS - INDIRECT POINTERS TO PROCEDURES AND COMMON
- . ENTRY CONTROL BLOCKS
- . POINTED TO BY LB

(111) STACK AREA:
- .1 PER INVOCATION
- . CALLER'S STATE
- . ARGUMENT LIST
- . FORTRAN TEMPORARIES
- . POINTED TO BY SB

Base Register

There are four base registers associated with 'Procedure Call' called by a user:

PB - Pointed to the beginning of the procedure segment.

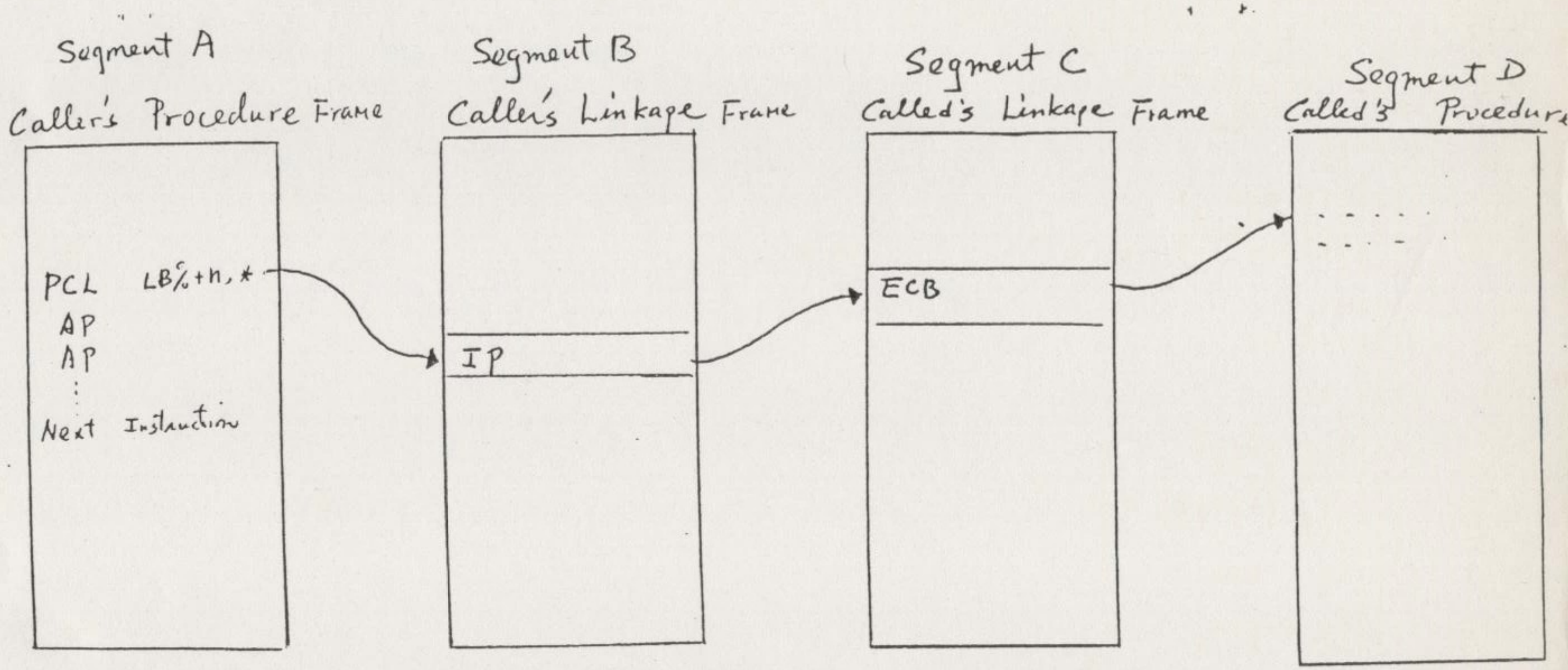
LB - Pointed to a location '400 location before the beginning of the linkage area.

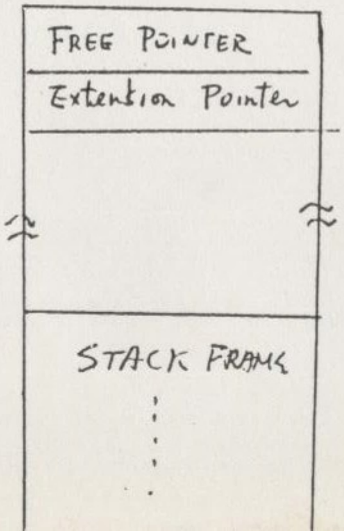
SB - Pointed to the current stack frame.

XB - Extra base register for users to use.

Direct Entrance Call

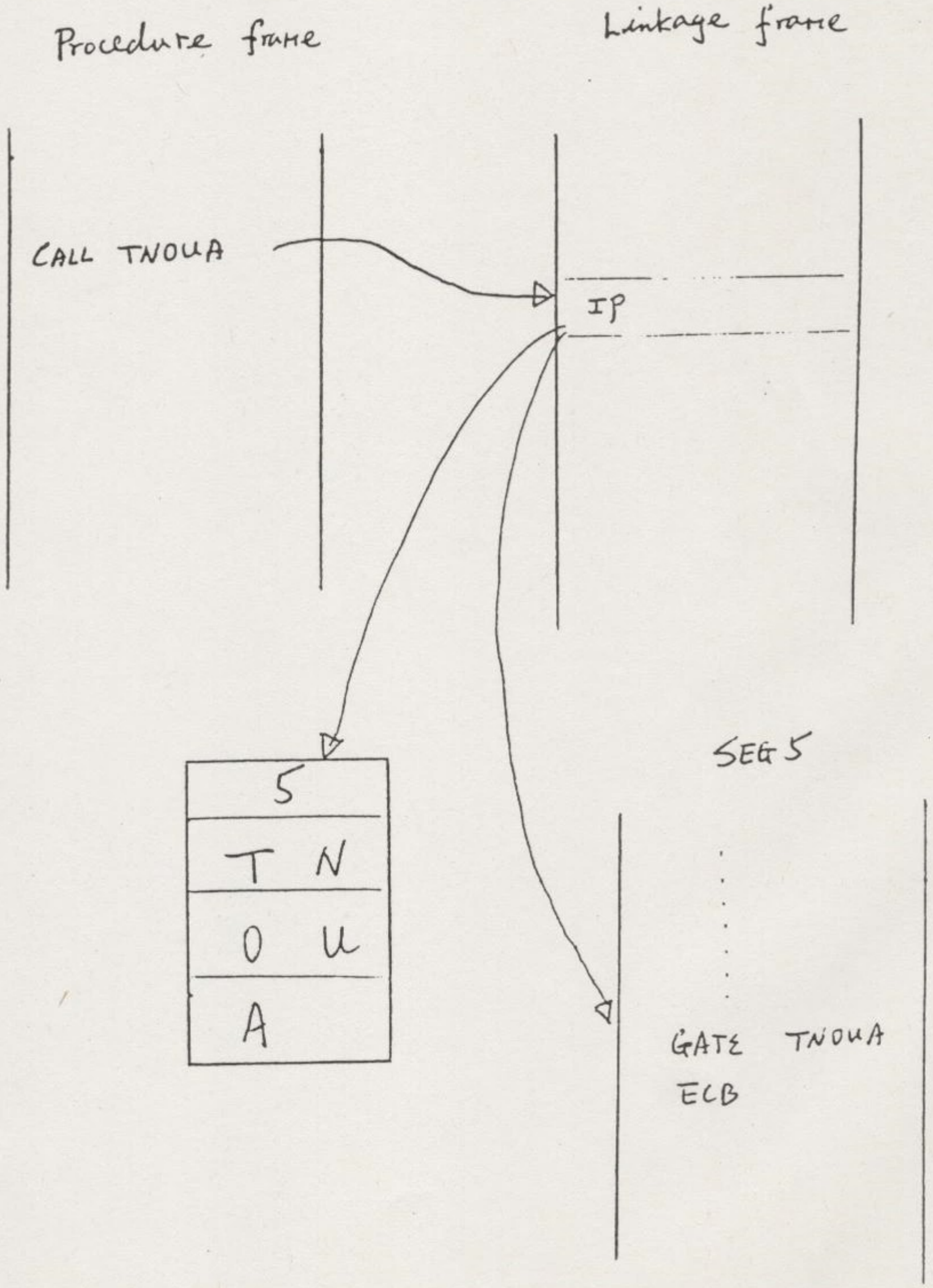
A procedure call to a routine which is implemented in the operating system but is gated through is called a Direct Entrance Call. See Figure ____.





Procedure CALL

Example of A Direct Entrance Call



PROCESS EXCHANGE & SCHEDULING

Process Exchange

One of the operating system's responsibilities is to decide which process is scheduled to run next and set up the necessary steps for this process. The first step is done by software modules, such as SCHED, PABORT. The latter step is done by hardware/firmware, and the procedure is called process exchange.

The data bases for Process Exchange are:

READY List

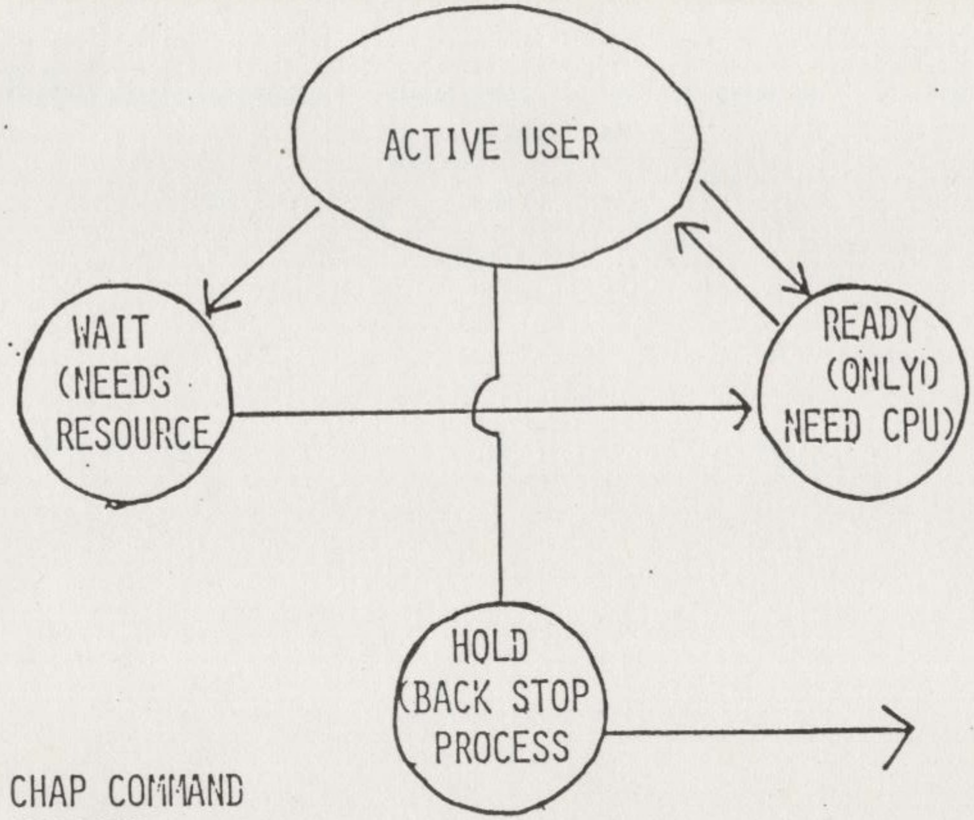
PCB (Process Control Block)

WAIT List

The root of Process Exchange is the Dispatcher, which is done in hardware.

The Dispatcher assigns a register set to the process which is scheduled to run and turn on the timer. It also scans the READY List looking for the process on the list.

(SET BY ELIG 3/10 SEC)



READY LIST PRIORITY

0	CLOCK PROCESS
1	AMLC
2	SMLC
3	MPC,MP2
8	SUPER-PROCESS 4
9	USER LEVEL 3
10	USER LEVEL 2
11	USER LEVEL 1
12	USER LEVEL 0
13	

BACKSTOP PROCESS

CHAP COMMAND CHANGES

WHEN HOLD

HIPRI, - WHEN C.R. PUT IN HIPRI TO GIVE FULL ELIG 3/10 SEC NEXT TIME USED

ELIG - USER +5 RUNOUT - STILL RDY 2 SEC TIME HAS RUN OUT

LOWPRI - 5 LEVELS WHEN 2 SEC TIME EXHAUSTED

HIPRI — NEW ELIG 1/3
+5 -2

ELIGQ — 2 SEC DEFAULT

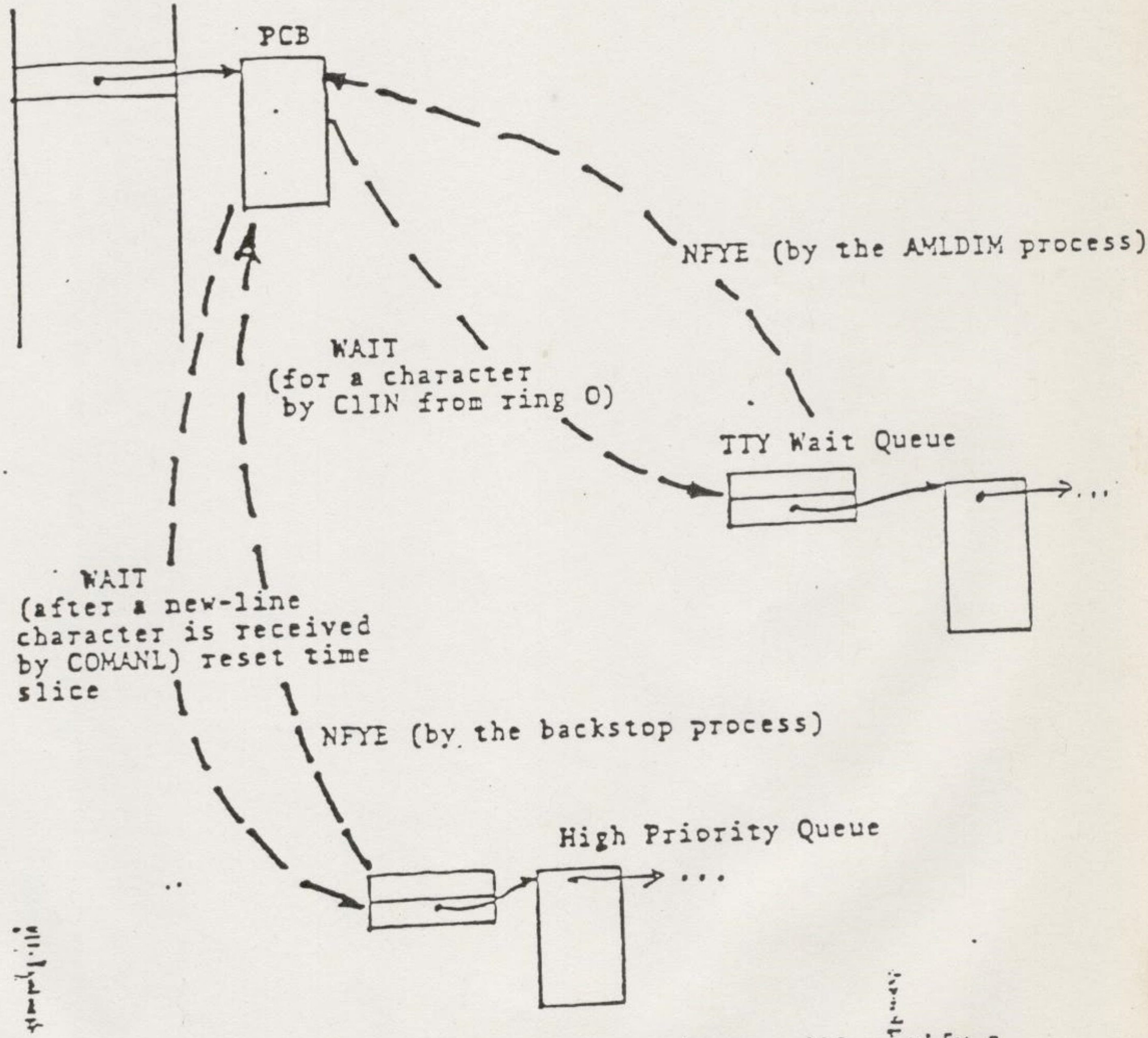
LOWPRI

4
3
2
1
0

WHEN COMING OUT, RESTORE 2 SEC TIME FOR ELIG

SCHEDULING

"INTERACTIVE USER" CYCLE



When there is no process ready, the backstop will notify a process on the high priority queue if there are any.

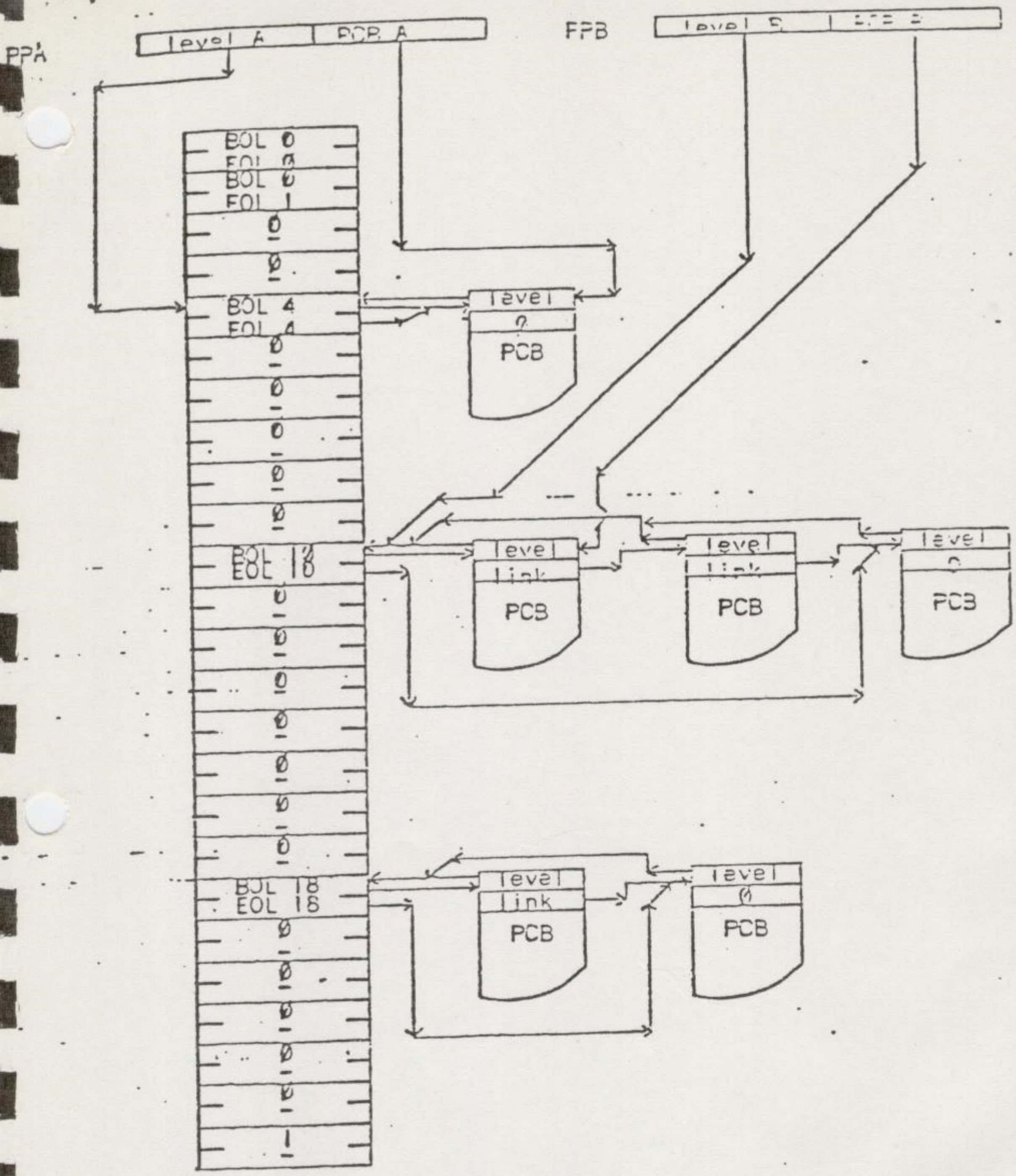
OBJECTIVES OF PRIMOS 4 SCHEDULING POLICY

- FAST RESPONSE TO INTERACTIVE USERS
- AVOID THRASHING
- SOME PROCESSING ON GRINDERS

THE PRINCIPLE CONSEQUENCE OF THE PROCESS PER USER ORGANISATION OF PRIMOS IV AT REV 14 IS THAT THIS POLICY IS NO LONGER IMPLEMENTED BY CHARACTERISING THE "STATE" OF A USER BY A NUMBER ASSOCIATED WITH THE PROCESS, BUT BY WHICH QUEUE - READY LIST OR WAIT LIST, THE PROCESS CONTROL BLOCK IS THREADED ON.

SCHEDULING POLICY IS THEN EMBODIED IN THE STRUCTURE OF NOTIFY AND WAIT INSTRUCTIONS THAT, ON CERTAIN EVENTS, (E.G. END OF TIME SLICE) ARE USED TO PUT THE PCB ONTO AN APPROPRIATE QUEUE.

- A PROCESS MAY BE NOTIFIED TO THE BEGINNING OR END OF THE READY QUEUE
- A PROCESS MAY WAIT ON ANY OF SEVERAL SEMAPHORES
- A PROCESS MAY BE REQUESTED TO REMOVE ITSELF FROM THE READY QUEUE TO A WAIT QUEUE BY SETTING ITS ABORT FLAG



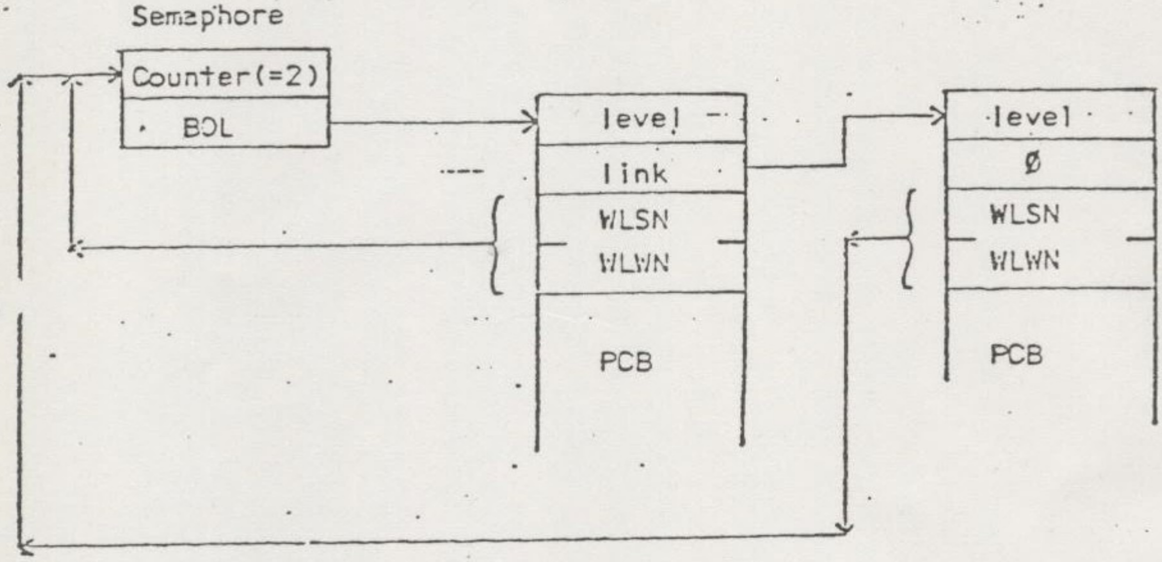
Ready List: All pointers are 16-bit word number pointers within the PCB segment. The segment number is contained in the high portion of the OWNER pointer within each register set.

All PCB start addresses must be even (bit 16 = Ø). The end of the ready list is marked with a BOL entry = 1.

FIGURE 1.

MLG 4 Sept 1975

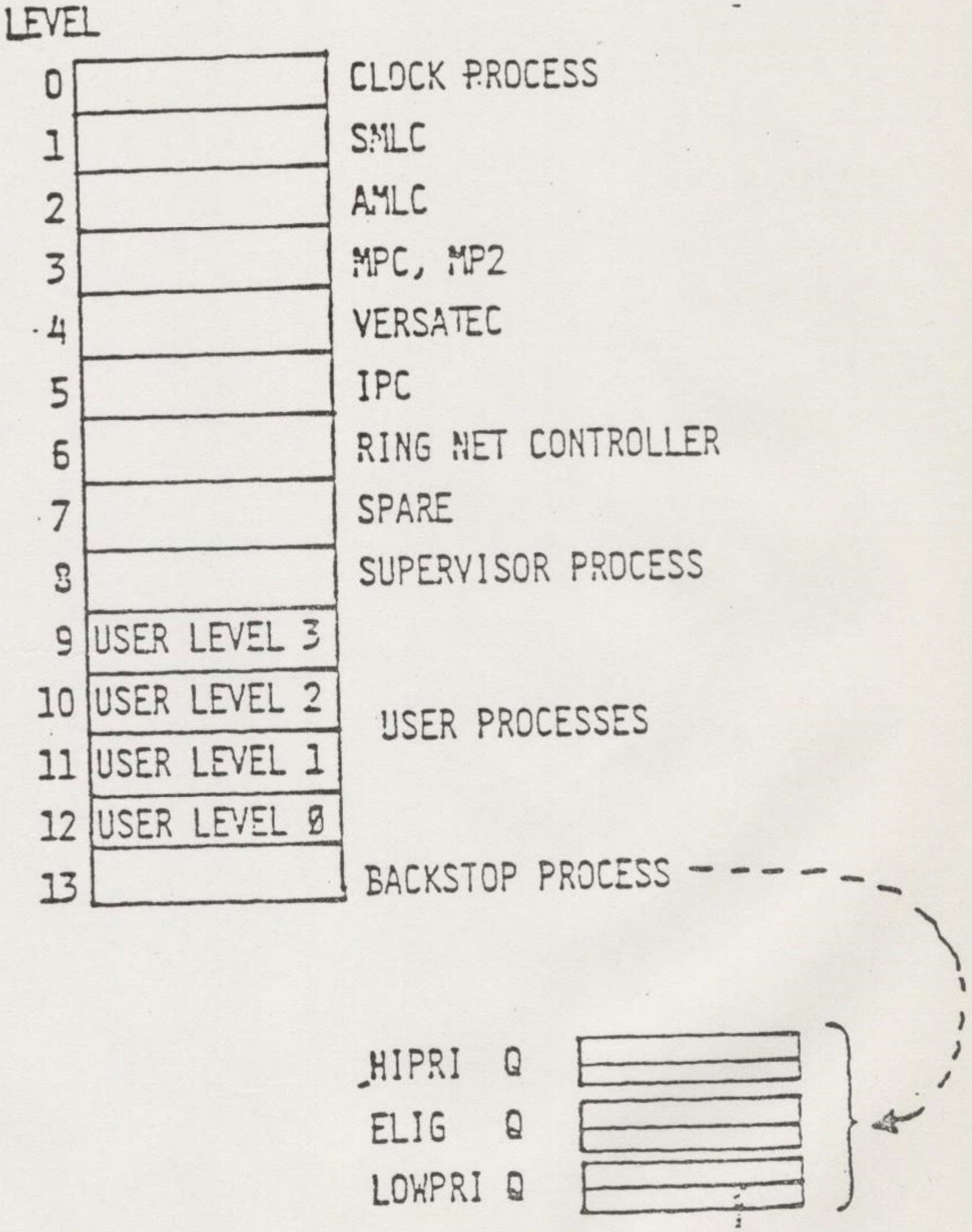
WAIT LIST STRUCTURE



QUEVING IS PRIORITY ORDER WITH FIFO FOR EQUAL PRIORITY

Figure 2.

REV. 15 READY LIST:



LEVEL		
0		CLOCK PROCESS
1		SMLC
2		AMLC
3		MPC, MP2
4		VERSATEC
5		IPC
6		RING NET CONTROLLER
7		SPARE
8		SUPERVISOR PROCESS
9	USER LEVEL 3	USER PROCESSES
10	USER LEVEL 2	
11	USER LEVEL 1	
12	USER LEVEL Ø	
13		BACKSTOP PROCESS

REV. 16 READY LIST:

LEVEL		
0		CLOCK PROCESS
1		AMLC
2		SMLC
3		MPC, MP2
4		VERSATEC
5		IPC
6		RING NET CONTROLLER
7		SPARE
8		SUPERVISOR PROCESS
9	USER LEVEL 3	USER PROCESSES
10	USER LEVEL 2	
11	USER LEVEL 1	
12	USER LEVEL Ø	
13		BACKSTOP PROCESS

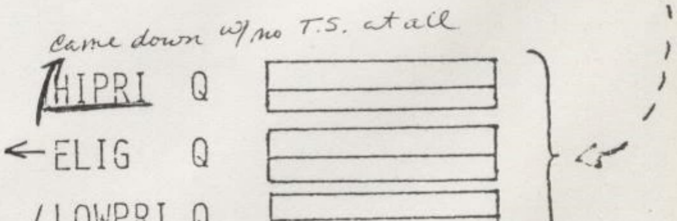
always "curr user" or "ready" state

2 SEC. TIME SLICE
INTERRUPTED EVERY 1/3 SEC.
TO GIVE LOW-PRI USERS A
SHOT @ CPU.

(BAL. BET. COMPUTE-BOUND USERS [HI-PRI]
AND GOOD RESPONSE FOR I/O BOUND [LO-PRI])

IF 2-SEC SLICE NOT USED UP @ 1/3 SEC.
INTERRUPT, GOES TO ELIGTS Q.

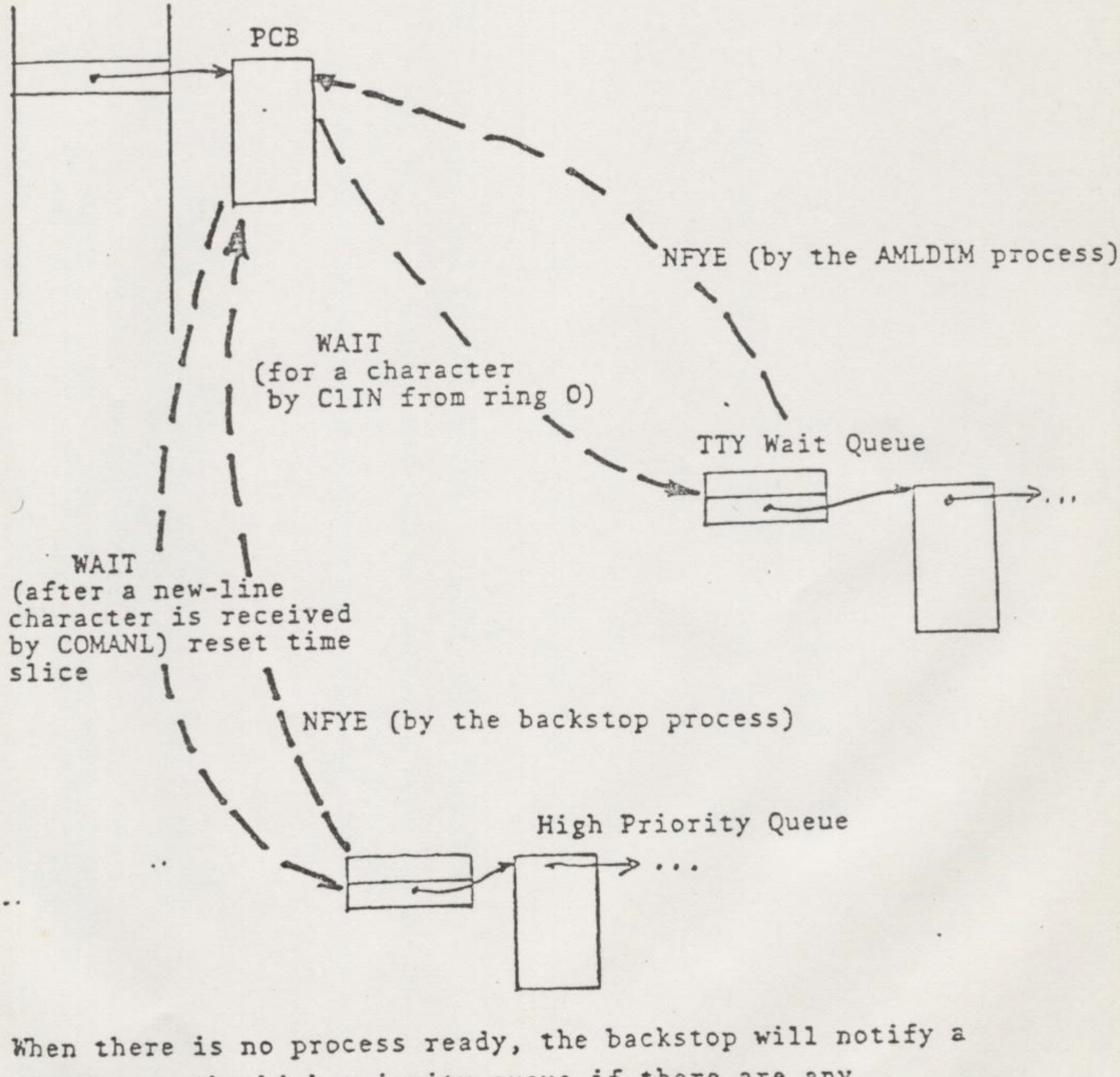
came down w/no T.S. at all



T.S. and elig TS run
out, then you get both
new & come here

SCHEDULING

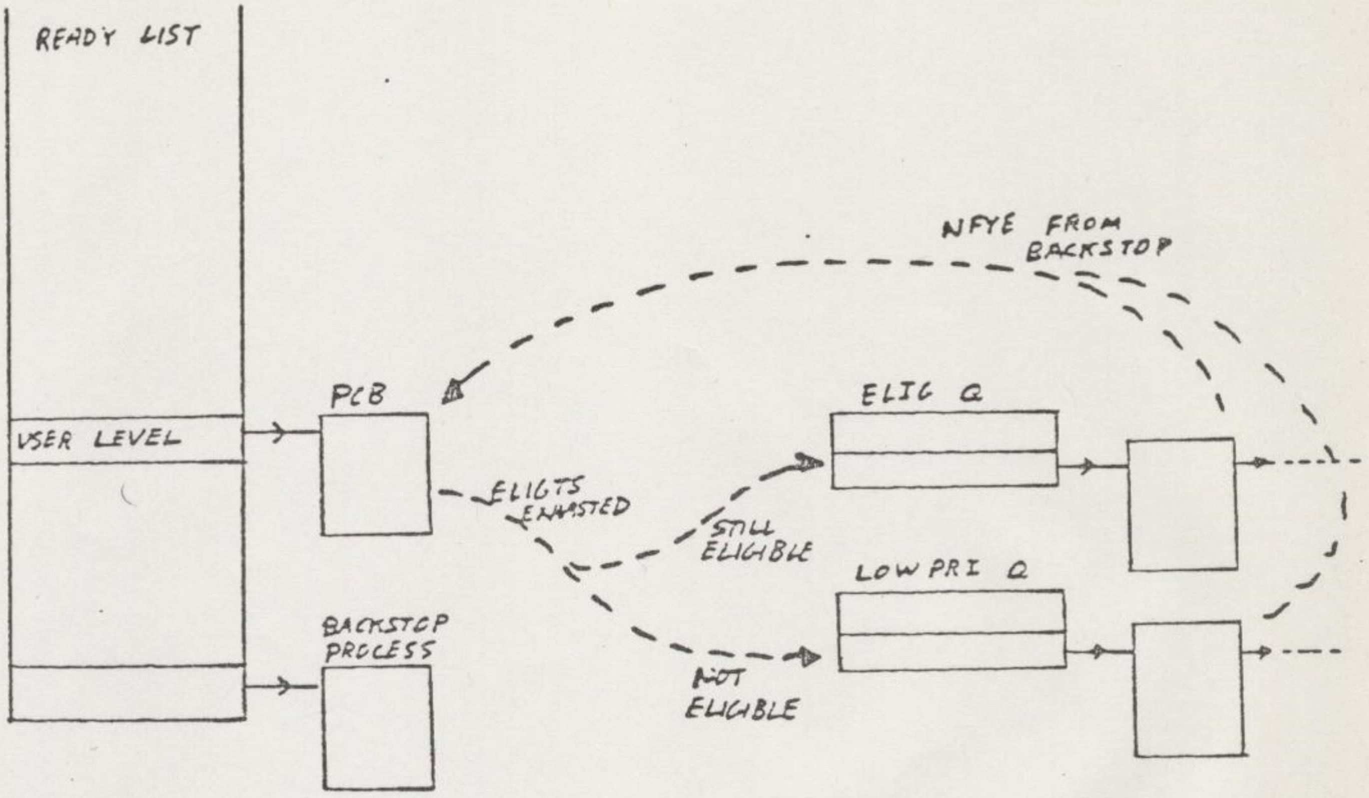
"INTERACTIVE USER" CYCLE



When there is no process ready, the backstop will notify a process on the high priority queue if there are any.

SCHEDULING

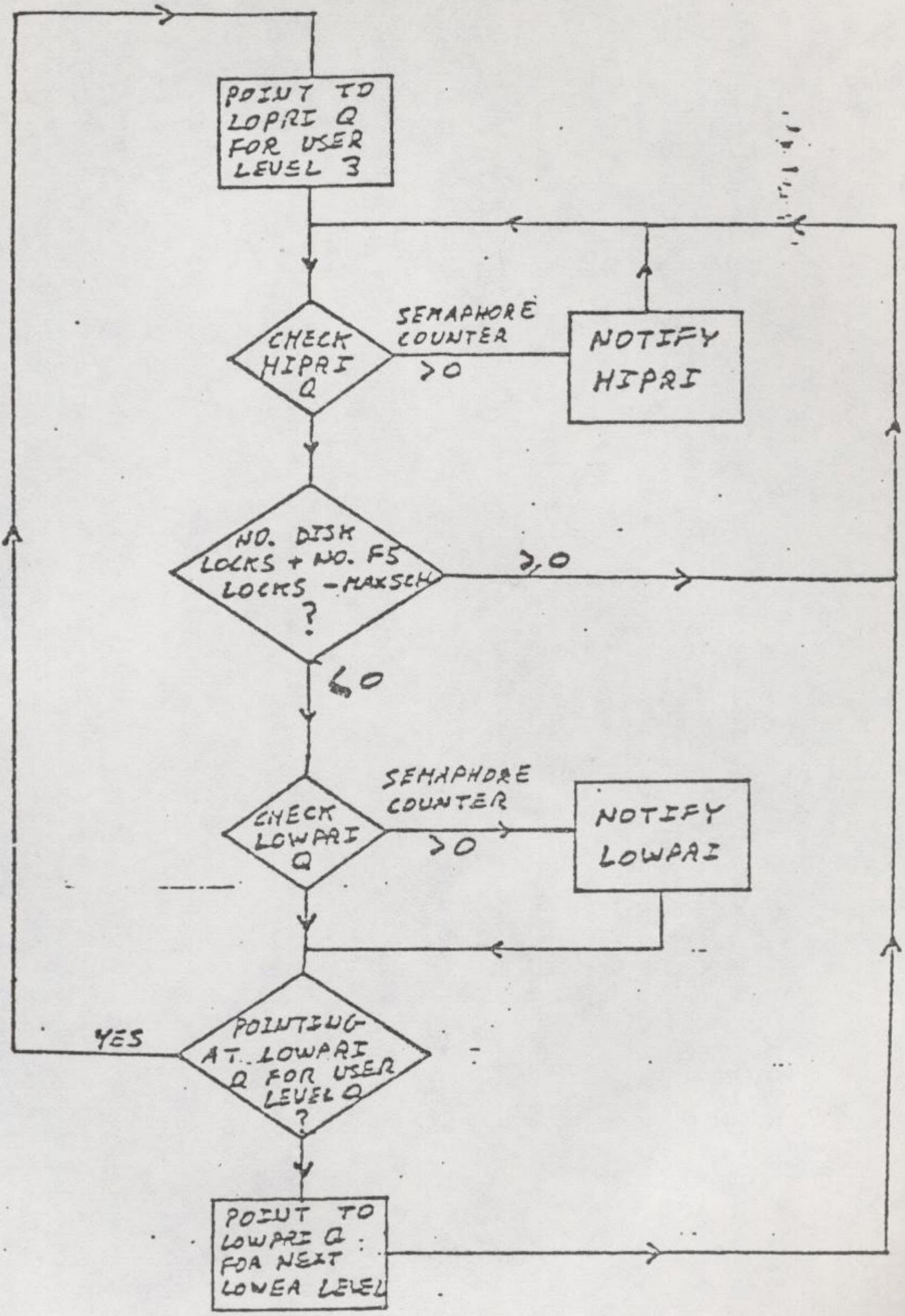
"COMPUTE BOUND" CYCLE



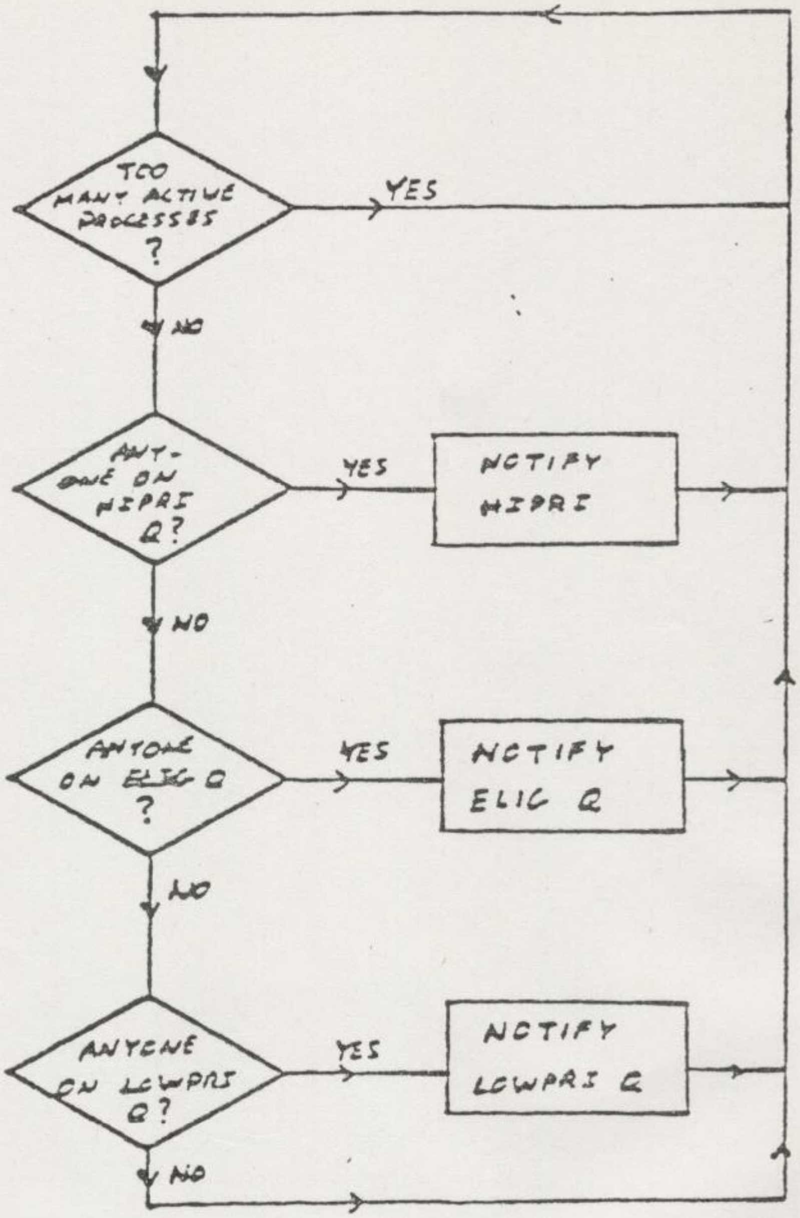
WHEN ELIGTS (DEFAULT 1/3 SEC) IS UP, PROCESS WAITS ON ELIGIBILITY Q IF ITS TIMESLICE (DEFAULT 2 SECONDS) IS NOT EXHAUSTED. OTHERWISE WAITS ON LOW PRIORITY Q

ELIGTS IS RESET ON NOTIFY FROM ELIGIBILITY Q. TIMESLICE IS RESET ON NOTIFY FROM LOW PRIORITY Q

BACKSTOP PROCESS REV 14.0 (simplified)


POINT TO LOPRI Q FOR USER LEVEL 3
CHECK HIPRI Q
SEMAPHORE COUNTER
>0
NOTIFY HIPRI
NO. DISK LOCKS + NO. FS LOCKS - MAXSCH ?
≥0
<0
CHECK LOWPRI Q
SEMAPHORE COUNTER
>0
NOTIFY LOWPRI
POINTING AT LOWPRI Q FOR USER LEVEL 0 ?
YES
POINT TO LOWPRI Q FOR NEXT LOWER LEVEL

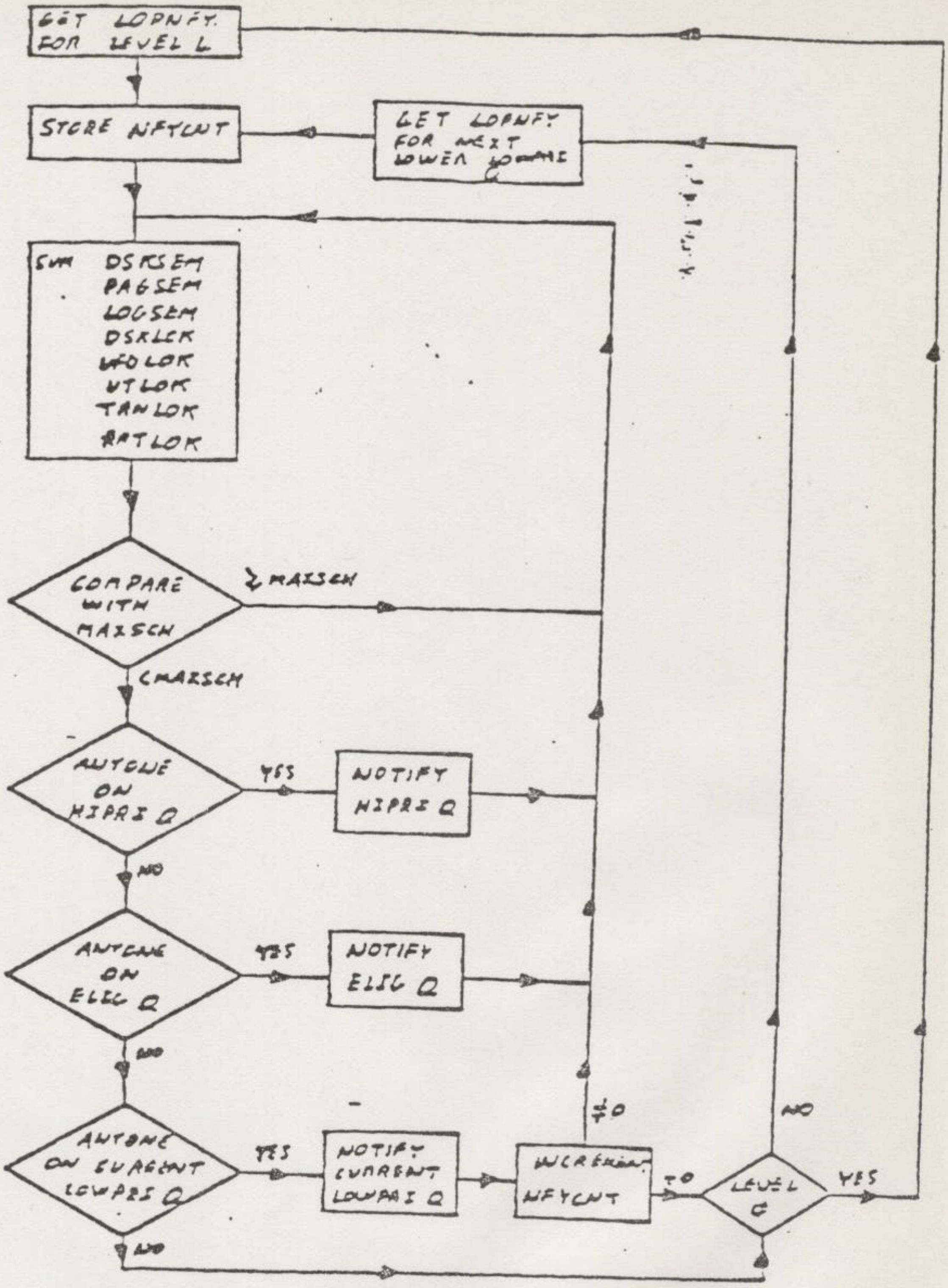
BACKSTOP PROCESS (SIMPLIFIED) Rev. 15



ACTIVE PROCESSES DEFINED AS THOSE ON FS LOCKS, DSKLCK, PAGLCK. A PARAMETER CALLED MAXSCH IS USED TO CONTROL THE NO. OF ACTIVE PROCESSES. THIS NOW CONTROLS INTERACTIVE USERS AS WELL AS GRINDERS.

NOTE: 'QUITS' CAN TAKE A LONG TIME TO RESPOND IF PROCESS IS ON LOWPRI Q.

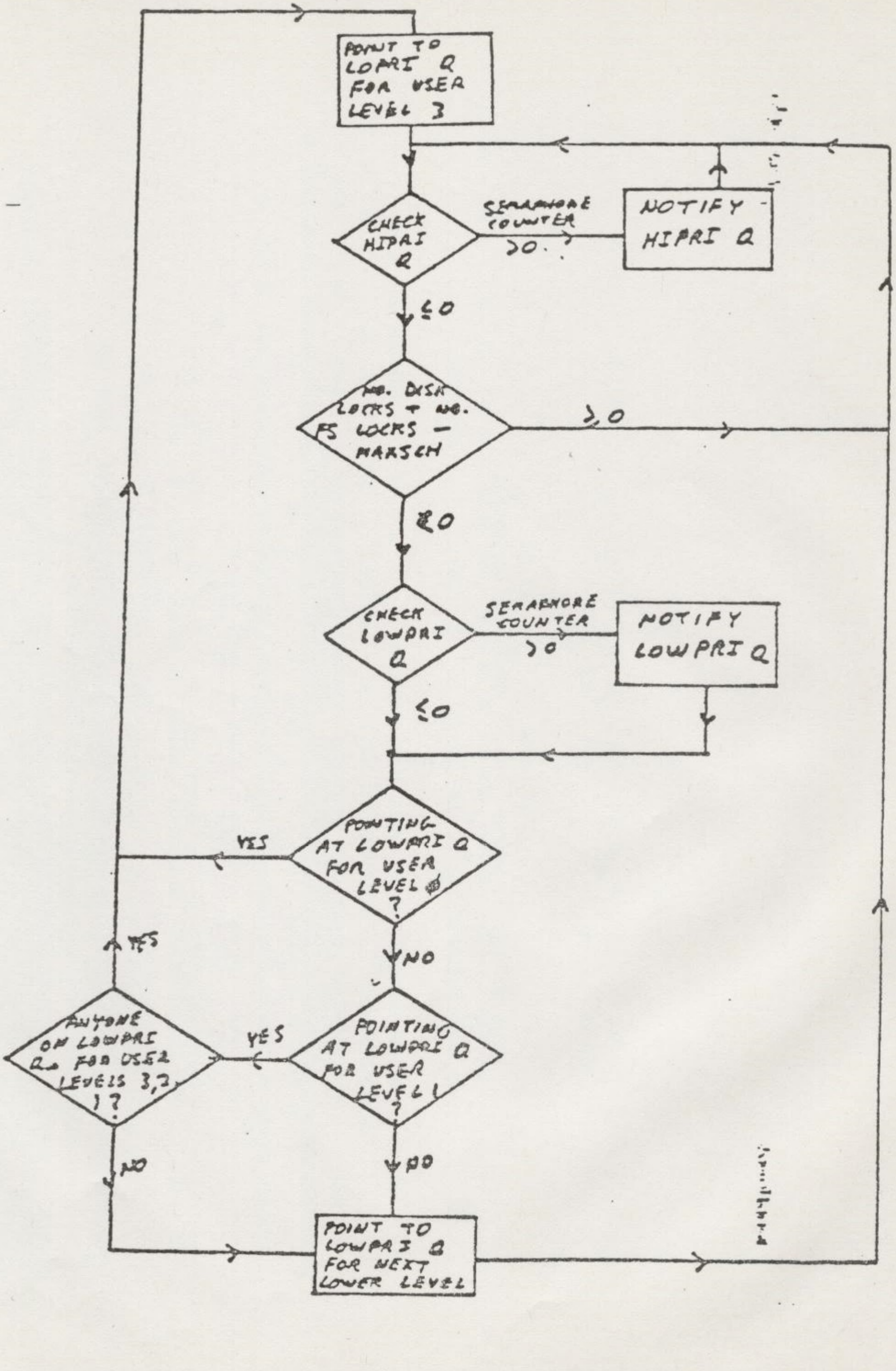
BACKSTOP PROCESS: REV 15



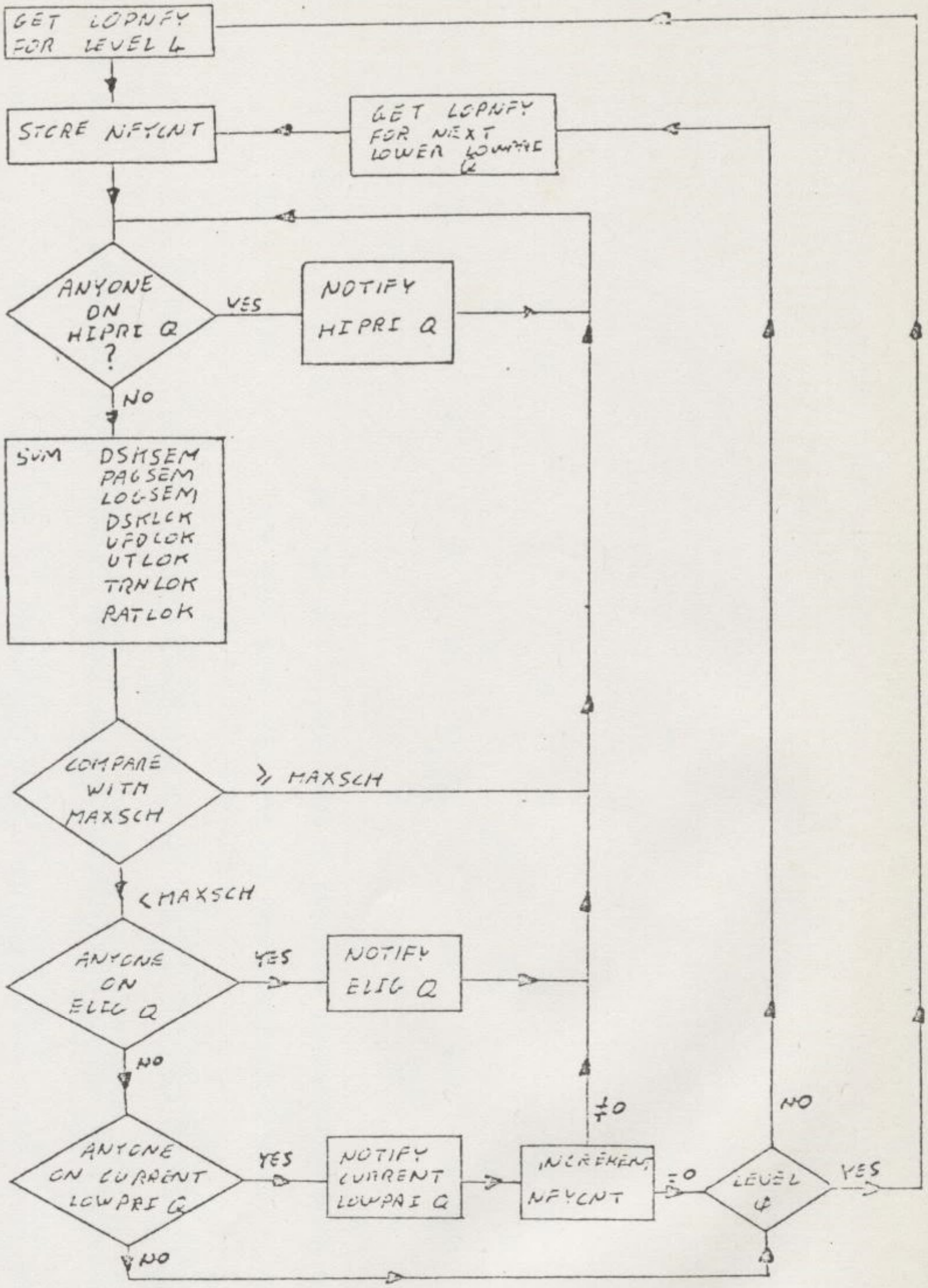
LOPNFY ALLOWS 16 NOTIFIES ON LEVEL 4 LOWPRI Q
8 NOTIFIES ON LEVEL 3 LOWPRI Q
4 NOTIFIES ON LEVEL 2 LOWPRI Q
2 NOTIFIES ON LEVEL 1 LOWPRI Q
1 NOTIFIES ON LEVEL Ø LOWPRI Q

NFYCNT CONTAINS CURRENT NO. OF NOTIFIES ON CURRENT LOWPRI Q. WHEN NO ONE IS ON THE CURRENT LOWPRI LEVEL, GO TO NEXT LEVEL IRRESPECTIVE OF NFYCNT

PRIMOS IV REV 14.1 BACKSTOP PROCESS (simplified)


POINT TO LOPRI Q FOR USER LEVEL 3
CHECK HIPRI Q
SEMAPHORE COUNTER >0
NOTIFY HIPRI Q
≤0
NO. DISK LOCKS + NO. FS LOCKS − MAXSCH
>0
≤0
CHECK LOWPRI Q
SEMAPHORE COUNTER >0
NOTIFY LOWPRI Q
≤0
POINTING AT LOWPRI Q FOR USER LEVEL ∅ ?
YES
NO
YES
ANYONE ON LOWPRI Q FOR USER LEVELS 3,2,1?
YES
POINTING AT LOWPRI Q FOR USER LEVEL 1 ?
NO
NO
POINT TO LOWPRI Q FOR NEXT LOWER LEVEL

BACKSTOP PROCESS: Rev 16



LOPNFY ALLOWS 16 NOTIFIES ON LEVEL 4 LOWPRI Q
8 NOTIFIES ON LEVEL 3 LOWPRI Q
4 NOTIFIES ON LEVEL 2 LOWPRI Q
2 NOTIFIES ON LEVEL 1 LOWPRI Q
1 NOTIFIES ON LEVEL Ø LOWPRI Q

NFYCNT CONTAINS CURRENT NO. OF NOTIFIES ON CURRENT LOWPRI Q.
WHEN NO ONE IS ON THE CURRENT LOWPRI LEVEL, GO TO NEXT LEVEL IRRESPECTIVE OF NFYCNT

MAXSCH COMMAND:

USED TO SET THE SCHEDULING CONSTANT MAXSCH FROM SYSTEM TERMINAL

MAXSCH <N>

DEFAULT SHOULD BE 3.

NOTE THAT MAXSCH IS CALCULATED AT CONFIG TIME ACCORDING TO AVAILABLE MEMORY:

MEMORY	MAXSCH
64K WORDS	0
96	1
128	2
160	3
⋮	⋮

ELIGTS COMMAND:

USED TO MODIFY THE ELIGIBILITY TIMESLICE FROM THE SYSTEM TERMINAL

ELIGTS <N> , WHERE N = NEW VALUE IN TENTHS OF A SECOND

DEFAULTS TO 3/10 SECOND.

CHAP COMMAND:

AS AT REV.14. CAN BE USED TO CHANGE PRIORITY AND TIMESLICE ON A PER USER BASIS. NOTE THAT DEFAULT TIMESLICE IS 2 SECONDS.

Rev 17

SCHEDULING

- MAXSCH DEFAULTS TO 4 FOR SYSTEMS WITH 448KB OR MORE
- BACKSTOP KNOWS ABOUT THE NEW DISK QUEUING MECHANISM WHEN CALCULATING THE NUMBER OF ACTIVE PROCESSES
- WITH MULTIPLE DRIVES, MAY BE POSSIBLE TO IMPROVE SYSTEM THROUGHPUT BY RAISING MAXSCH

COMMAND LINE PROCESSOR

Command Line Processor

In Revision 16 and prior to it, the module DOSSUB is 'the' command processor. The commands are categorized into two groups:

internal and external commands

All internal commands codes reside in DOSSUB. All external commands' run images live in an UFD called CMDNCØ.

In Revision 17, a major change occurs in the command line processor -- call it New Command Line Processor. It has two distinct modes:

static mode and recursive mode

Currently, all user's programs and all external commands are executed in static mode. PRIMOS codes, internal commands, as well as the condition mechanism, are executed in recursive mode.

There are four groups of commands in Revision 17; they are:

- Old Ring 3 internal commands:

  START and RESTORE

- New Ring 3 internal commands:

  ABBREV, RLS, REN,
  DMSTK, RDY

- Ring O internal commands:

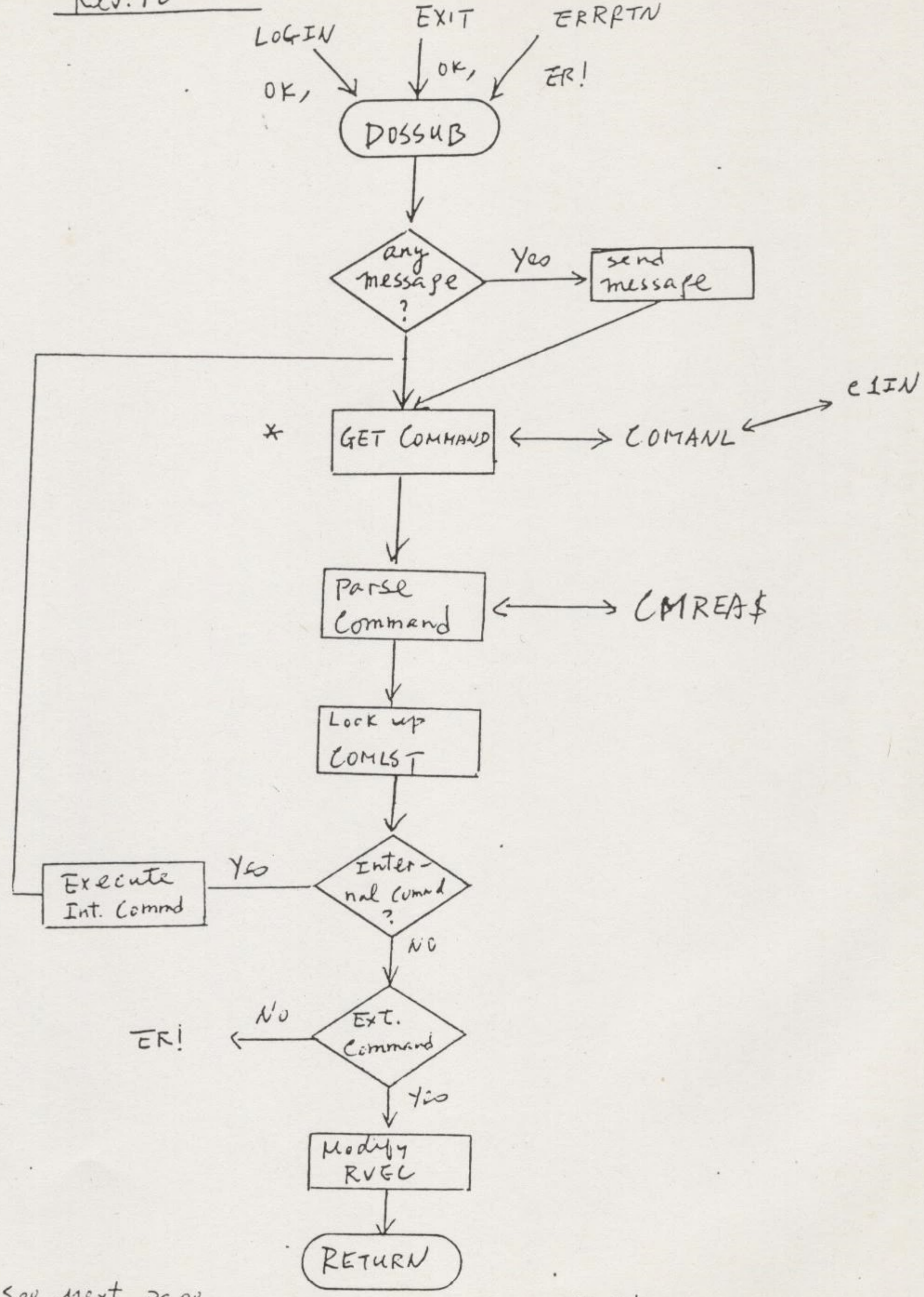
  DOSSUB's internal commands

- External commands:

  Such as utility programs,
  compilers, and external
  commands installed by users.

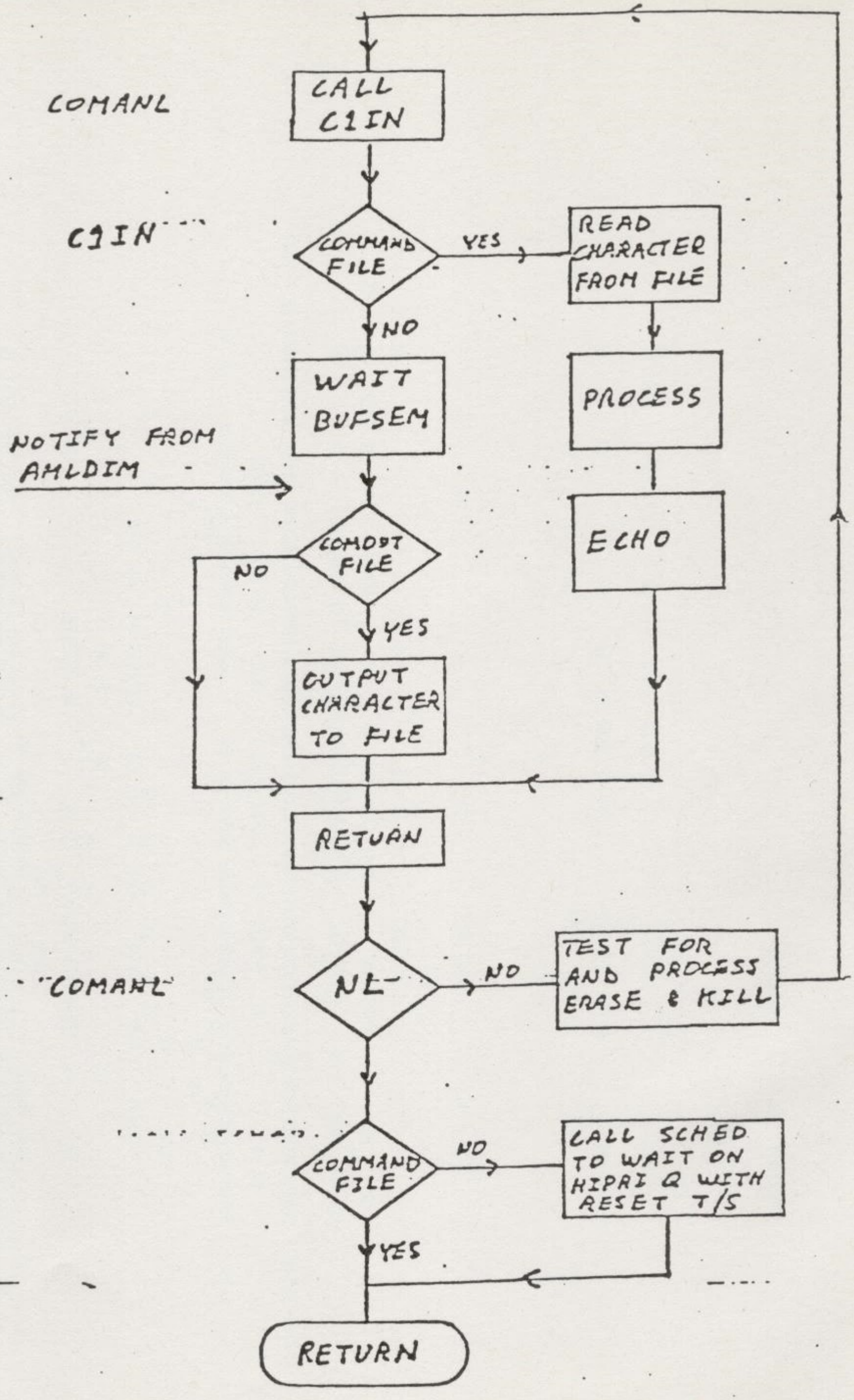
New Command Line Processor is illustrated in Figure ____.

Rev. 16



* See next page

READ A COMMAND LINE



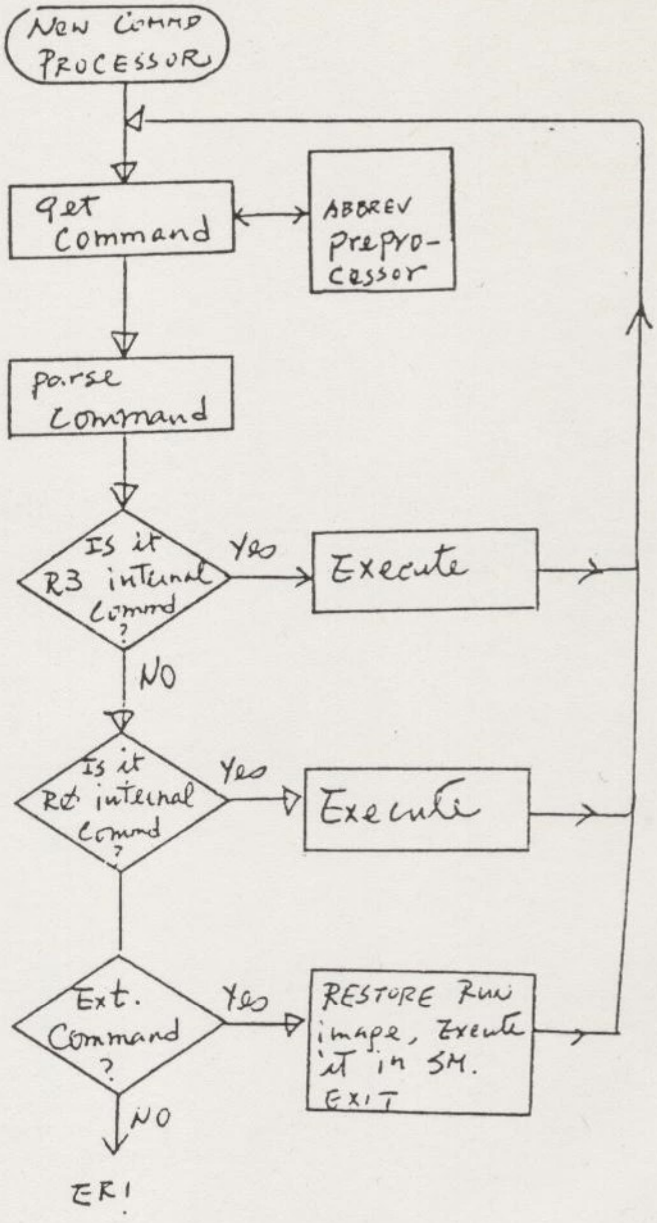
Rev. 17

CL$GET

Yes/ DOSSUB

Yes/INVKSM_

ERRTN



DEBUGGING

DBG - SOURCE LEVEL DEBUGGER

(I) Overview

(1) Addressing Modes: DBG operates on programs which execute in either 64V or 32I modes. The debugger itself executes in 64V mode.

(2) Languages Supported: FORTRAN-74,FORTRAN-77,PL/1,PL/P. COBOL support is planned.

(3) Memory Requirement: The debugger's procedure part (which is shared) occupies 3 segments. Per user information requires a fixed amount of space includes common area and linkage text. This occupies about 48K words. Per user space of variable length includes stack space (at leas 16K words) and symbol table space. All symbol table storage is allocate dynamically.

(4) Central Processor: The DBG runs on any CPU capable of generating 64V addressing mode. Presently, this includes PRIME 350,400,450,500,550, 650 and 750 processors.

(II) PROCEDURE OF CALLING DBG

(1) Program Compilation

The user must inform the compiler that he/she later intends to use DBG. This is done by including the '-debug' parameter as one of the compile-time options on the command line.

For example, to compile 'myprogram' with the FORTRAN compiler for later use of DBG, one enters:

```
OK, FTN MYPROGRAM -64V -DEBUG
```

Inclusion of the '-DEBUG' option causes the compiler to output the information necessary for the debugger to recognize and manipulate program units, symbols and statements.

(2) Program loading

Programs which are compiled with '-DEBUG' option are loaded in the same way as those which are not, in other words, the user experience no change in program loading.

(3) Invoking and Terminating DBG

The debugger is invoked at PRIMOS level by 'DBG' command followed by the name of the SEG file containing the program to be debugged.

For example, to debug the '#myprogram':

```
OK, DBG #MYPROGRAM
**DBG**  revision 17.0a  (06-February-1979)
>
```

With this command, the debugger is entered. It reads the program and symbol table from the SEG file into memory and prints an ID message as well as a prompt sign >. The debugger's command may be entere When the 'quit' command is entered, the control is returned to PRIMOS command level.

Example:

```
> QUIT
OK,
```

(4) User Program Control

Control is initially passed to DBG from PRIMOS when the debugger is invoked. Control passed from DBG to user's program when

. the user uses RESTART or CONTINUE command to restart or continue program execution.
. the user gives one of the single-step commands, such as STEP, STEPI In, or OUT.
. the user CALLs a subroutine contained within the user program, or when the evaluation of an expression involves a user-defined function.

Control returns to DBG when

. the user program encounters a breakpoint previously set by user.
. the program completes execution of the number of statements implied or expressed in a single-step command,
. the main program returns, or any program unit stops, pauses, calls EXIT or calls ERRPR$ to return to PRIMOS command level,
. in entry trace mode, whenever a procedure is called or returns,
. in statement and/or value trace modes, whenever a procedure is call or returnes, and prior to the execution of each statement,
. a user's subroutine or function returns from a call made fron DBG on behalf of the user,
. when the user depresses the 'quit' key at his/her terminal, provide the user program has no handler or the QUIT$ condition.

List of Debugger's Commands

RESTART

CONTINUE

GOTO

MAIN

BREAKPOINT

TRACEPOINT

CLEAR

CLEARALL

LIST

LISTALL

TYPE

LET

ARGUMENTS

STEP

STEPIN

IN

OUT

ETRACE

STRACE

TRACEBACK

WATCH

WATCHLIST

UNWATCH

VTRACE

PRIMOS BUILD

This section will be devoted to PRIMOS build. It is necessary to build PRIMOS when you

- Modify one of the operating system codes.

- Install a Ring 0 or Ring 3 internal command.

- Install a Direct Entrance Call.

The PRI400 directory is where all the source programs and the corresponding object codes reside. PRIME supplies the source program so that user may modify or add a module in the operating system.

There will be a demonstration for PRIMOS build.

** Listing of C_ALL **

```
/* ALL,  PRI400,  BIN-CMW,  03/14/79
/*    COMPILE AND LOAD ALL SOURCES FOR PRIMOS AND ITS UTILITIES
/*    COPYRIGHT 1979, PRIME COMPUTER INC., WELLESLEY,MA  02150
/*
COMO O_ALL
/*
CO C_COMO.OFF 20
/*
CO C_VPSD 20
/*
/*
CO C_PRMLD 20        /* to build the preloader run file - PRIMOS
/*
/*
CO C_MAPGEN 20       /* Generate MAPGEN program for PRIMOS - *MAPGEN
/*
/*
CO C_KS 20           /* Compile and/or assemble source progrms in KS
/*
/*
CO C_FS 20                         /* compile or assemble source programs in FS
/*
/*
CO C_NS 20                         /* compile or assemble source programs in NS
/*
/*
CO C_CS 20                         /* compile or assemble source programs in CS
/*
/*
CO C_SE 20                         /* compile or assemble source programs in SE
/*
/*
CO C_R3S 20                         /* compile or assemble source programs in R3S
/*
/*
CO C_PLPLIB 20                      /* compile or assemble source programs in PLPL
/*
CO C_COMO.ON 20
/*
CO C_R3LOAD 20       /* Load ring3 object codes and build PR0013, PR5002
CO C_LOAD 20         /* Load ring0 object codes, build run files PR000 -- PR
/*
/*
COMO -END
CO -END
```

```
OOOOO  PPPP   EEEEE  RRRR    AAA   TTTTT  IIIII  OOOOO  N   N   SSS
O   O  P   P  E      R   R  A   A    T      I    O   O  NN  N  S
O   O  PPPP   EEE    RRRR   AAAAA    T      I    O   O  N N N   SSS
O   O  P      E      R R    A   A    T      I    O   O  N  NN      S
OOOOO  P      EEEEE  R  R   A   A    T    IIIII  OOOOO  N   N   SSS
```

DEVICE NUMBERS

To build or modify a partition you run a command called MAKE. In Appendix A, there is an example of MAKE being used to change two smaller partitons into a larger partition. However before you can run make, you must calculate the physical device number for that partition. A physical device number is a six diget octal number that tells the system how large the partitonis and precisely where it is located on the disk pack. Below is an example of a physical device number.

ØØ	Ø46	Ø
$\frac{\text{starting head no.}}{2}$	controller address	drive unit no. x 2 or drive unit no. x 2 + 1

For every physical device number, ther is also a logical device number. A logical device number is an octal number assigned to a partition during startup. The first partition added to the system is logical device 1, the next partition is logical device 2, etc.

PHYSICAL DEVICE NUMBER

NUMBER OF SURFACES

BEGINNING HEAD NUMBER

	0	2	4	6	8	10	12	14	16	18
1	------	------	020061	------	------	------	------	------	------	110061
2	000460	010460	020460	030460	040460	050460	060460	070460	100460	------
3	------	010461	------	------	------	------	------	------	100461	------
4	001060	011060	021060	031060	041060	051060	061060	071060	------	------
5	001061	------	------	------	------	------	------	071061	------	------
6	001460	011460	021460	031460	041460	051460	061460	------	------	------
7	------	------	------	------	------	------	061461	------	------	------
8	002060	012060	022060	032060	042060	052060	------	------	------	------
9	------	------	------	------	------	052061	------	------	------	------
10	002460	012460	022460	032460	042460	------	------	------	------	------
11	------	------	------	------	042461	------	------	------	------	------
12	003060	013060	023060	033060	------	------	------	------	------	------
13	------	------	------	033061	------	------	------	------	------	------
14	003460	013460	023460	------	------	------	------	------	------	------
15	------	------	023461	------	------	------	------	------	------	------
16	004060	014060	------	------	------	------	------	------	------	------
17	------	014061	------	------	------	------	------	------	------	------
18	004460	------	------	------	------	------	------	------	------	------
19	004461	------	------	------	------	------	------	------	------	------

THIS TABLE CONTAIN ALL THE POSSIBLE PHYSICAL DEVICE NUMBERS FOR THE 40, 80, AND 300 MB DISK DRIVES. TO USE THE TABLE DECIDE HOW MANY DISK SURFACES ARE TO BE INCLUDED IN YOUR PARTITION AND WHAT HEAD NUMBER IS THE FIRST HEAD IN THE PARTITION. USING THIS INFORMATION LOOK UP THE PHYSICAL DEVICE NUMBER IN THE TABLE. IF THE PARTITION YOU DEFINE DOES NOT SHOW UP ON THIS TABLE, THAN IT IS NOT A LEGAL PARTITION. FOR EXAMPLE, ALL PARTITIONS MUST BEGIN ON AN EVEN HEAD NUMBER AND ONLY THE LAST PARTITION ON THE DISK PACK CAN HAVE AN ODD NUMBER OF SURFACES. THESE TWO RULES MUST BE OBEYED.

NOTE - THE PHYSICAL DEVICE NUMBERS IN THIS TABLE ASSUME THAT THE DISK PACK IS MOUNTED ON DISK DRIVE 0. TO FIND THE PHYSICAL DEVICE NUMBERS FOR DISK PACKS MOUNTED ON OTHER DRIVES, TAKE THE DISK DRIVE UNIT NUMBER, MULTIPLE IT BY 2, AND ADD IT TO THE PHYSICAL DEVICE NUMBER FROM THE TABLE. THIS SUM IS THE PHYSICAL DEVICE NUMBER.

BEWARE - PHYSICAL DEVICE NUMBERS 020061, 010461, AND 001061 ARE ONLY POSSIBLE ON A 40 OR 80 MB DISK DRIVE. ALSO NOTE THAT THE 40 AND 80 MB DISKS ONLY HAVE HEADS 0 THRU 4.

PHYSICAL DEVICE NUMBER - 2nd CONTROLLER

NUMBER OF SURFACES	BEGINNING HEAD NUMBER									
	0	2	4	6	8	10	12	14	16	18
1	------	------	020261	------	------	------	------	------	------	110261
2	000660	012660	022660	032660	042660	050660	060660	070660	100660	------
3	------	012661	------	------	------	------	------	------	100661	------
4	001260	011260	021260	031260	041260	051260	061260	071260	------	------
5	001261	------	------	------	------	------	------	071261	------	------
6	001660	011660	021660	031660	041660	051660	061660	------	------	------
7	------	------	------	------	------	------	061661	------	------	------
8	002260	012260	022260	032260	042260	052260	------	------	------	------
9	------	------	------	------	------	052261	------	------	------	------
10	002660	012660	022660	032660	042660	------	------	------	------	------
11	------	------	------	------	042661	------	------	------	------	------
12	003260	013260	023260	033260	------	------	------	------	------	------
13	------	------	------	033261	------	------	------	------	------	------
14	003660	013660	023660	------	------	------	------	------	------	------
15	------	------	023661	------	------	------	------	------	------	------
16	004260	014260	------	------	------	------	------	------	------	------
17	------	014261	------	------	------	------	------	------	------	------
18	004660	------	------	------	------	------	------	------	------	------
19	004661	------	------	------	------	------	------	------	------	------

THIS TABLE CONTAIN ALL THE POSSIBLE PHYSICAL DEVICE NUMBERS FOR THE 40, 80, AND 200 MB DISK DRIVES. TO USE THE TABLE DECIDE HOW MANY DISK SURFACES ARE TO BE INCLUDED IN YOUR PARTITION AND WHAT HEAD NUMBER IS THE FIRST HEAD IN THE PARTITION. USING THIS INFORMATION LOOK UP THE PHYSICAL DEVICE NUMBER IN THE TABLE. IF THE PARTITION YOU DEFINE DOES NOT SHOW UP ON THIS TABLE, THAN IT IS NOT A LEGAL PARTITION. FOR EXAMPLE, ALL PARTITIONS MUST BEGIN ON AN EVEN HEAD NUMBER AND ONLY THE LAST PARTITION ON THE DISK PACK CAN HAVE AN ODD NUMBER OF SURFACES. THESE TWO RULES MUST BE OBEYED.

NOTE - THE PHYSICAL DEVICE NUMBERS IN THIS TABLE ASSUME THAT THE DISK PACK IS MOUNTED ON DISK DRIVE 0. TO FIND THE PHYSICAL DEVICE NUMBERS FOR DISK PACKS MOUNTED ON OTHER DRIVES, TAKE THE DISK DRIVE UNIT NUMBER, MULTIPLE IT BY 2, AND ADD IT TO THE PHYSICAL DEVICE NUMBER FROM THE TABLE. THIS SUM IS THE PHYSICAL DEVICE NUMBER.

BEWARE - PHYSICAL DEVICE NUMBERS 020261, 012661, AND 001261 ARE ONLY POSSIBLE ON A 40 OR 80 MB DISK DRIVE. ALSO NOTE THAT THE 40 AND 80 MB DISKS ONLY HAVE HEADS 0 THRU 4.

PARTITIONING OF CARTRIDGE MODULE DEVICES

		First Controller	Second Controller
32 MB	Removable	61 (16 MB)	261 (16 MB)
	Non-removable	100061 (16 MB)	100261 (16 MB)
64 MB	Removable	61 (16 MB)	261 (16 MB)
	Non-removable	100460 (32 MB) 110061 (16 MB) or 100461 (48 MB)	100660 (32 MB) 110261 (16 MB) or 100661 (48 MB)
96 MB	Removable	61 (16 MB)	261 (16 MB)
	Non-removable	100460 (32 MB) 110460 (32 MB) 120061 (16 MB) or 101060 (64 MB) 120061 (16 MB) or 101061 (80 MB) or 100460 (32 MB) 110461 (48 MB)	100660 (32 MB) 110660 (32 MB) 120261 (16 MB) or 101260 (64 MB) 120261 (16 MB) or 101261 (80 MB) or 100660 (32 MB) 110661 (48 MB)

MAKE

* MAKE is a utility program used to create new partitions on a new pack or to change the size of existing partitions.

* In this example we shall recreate a 1062 partition into two partitions, 462 and 10462. There is one badspot on this pack: TRACK=603 HEAD=3. We shall run MAKE at user's terminal though it can be done at system consol

* First step is goto system consol and type:

```
    SHUTDN 1062
    DISK 462 10462
```

* Then, goto user terminal to run MAKE.

```
   OK, ASSIGN DISK 462
   OK, MAKE
   GO
   MAKE 16.8
   BUILDING NEW PARTITION.
   PHYSICAL DISK: 462
   40MB STORAGE MOD?NO
   SPLIT DISK?: NO
   DISK  FILE-RECORDS PAGE-RECORDS (DECIMAL)
   000462       14814              0
   PARAMETERS OK? YES
   PACK NAME?CLASS1
   BADSPOTS ON DISK? NO
   VIRGIN DISK? YES
   VERIFY DISK? YES
   FORMAT DISK? YES
   BEGINNING FORMAT
   FORMAT COMPLETED
   BEGINNING WRITE
   WRITE COMPLETE
   BEGINNING VERIFY
   DISK CREATED

   OK, UNAS DI 462
```

MAKE (II)

- MAKE partition with bad spots on it.
- We shall run Make on 10462 at user's terminal

```
OK, AS DI 10462
OK, MAKE
GO
MAKE 16.8
BUILDING NEW PARTITION.
PHYSICAL DISK: 10462
40MB STORAGE MOD?NO
SPLIT DISK?: NO
DISK  FILE-RECORDS PAGE-RECORDS (DECIMAL)
010462        14814            0
PARAMETERS OK? YES
PACK NAME?CLASS
BADSPOTS ON DISK? YES
TRACK = 607
HEAD = 3
TRACK = 0                      /* answer 0 to terminate BADSPOT list
HEAD = 0
TRACK    HEAD   OF BAD SPOT
   607      3

PARAMETERS OK? YES
VIRGIN DISK? YES
VERIFY DISK? YES
FORMAT DISK? YES
BEGINNING FORMAT
FORMAT COMPLETED
BEGINNING WRITE
WRITE COMPLETE
BEGINNING VERIFY
LOST RECORDS
DISK CREATED

OK, UNAS DI 10462
```

- Goto system consol issue the following commands to starts up the partitions:

```
DISK NOT 462 10462
ADDISK 462 10462
```

- NOTE: MAKE on paging surface can be done only under PRIMOS II
  The CMDNC0 and DOS are empty when a partition is made by MAKE.

  Extra step must be taken if you wish to modify the partition containing CMDNC0 and DOS. You must move these UFD's elsewhere before running MAKE.

MAINTENANCE

FIXRAT

- FIXRAT is an utility program that checks the PRIMOS file integrity on any partition. It reads every record in every file, directory and segment directory and checks its integrity. Should there be any inconsistency, FIXRAT prints out the discrepency with an error message.

- In this example, we shall run FIXRAT on 462.

- To run FIXRAT, first issue the following commands at system consol:

```
SHUTDN 462
DI 462
```

- Then proceed the following:

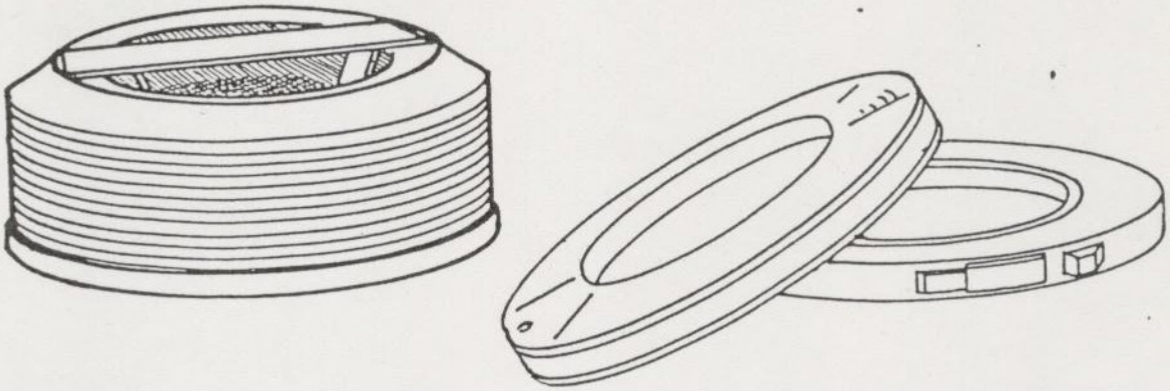
```
OK, AS DI 462
OK, FIXRAT
GO
FIXRAT 16.4
FIX DISK?  NO                  /* answer NO for the first time around
PHYSICAL DISK = 462
UFD COMPRESSION?YES

DISK PACK ID IS CLASS1
BEGIN MFD
  BEGIN CMDNC0
  END   CMDNC0           1
  BEGIN DOS
  END   DOS            1
  BEGIN SPOOLQ
  END   SPOOLQ          46
  BEGIN LEE
  END   LEE             15
  BEGIN XRI400
  END   XRI400         414
  BEGIN BEVERLY
  END   BEVERLY            12
  BEGIN MIKE
  END   MIKE           11
  BEGIN BOB
  END   BOB            31
  BEGIN ELTON
  END   ELTON             9
  BEGIN CHEN.2
  END   CHEN.2          23

END   MFD        569
RECORDS USED(DECIMAL)=          569
RECORDS LEFT=                 14245
DSKRAT OK
```

- FIXRAT done.
- UNASSIGN the disk
- Goto system consol and issue:

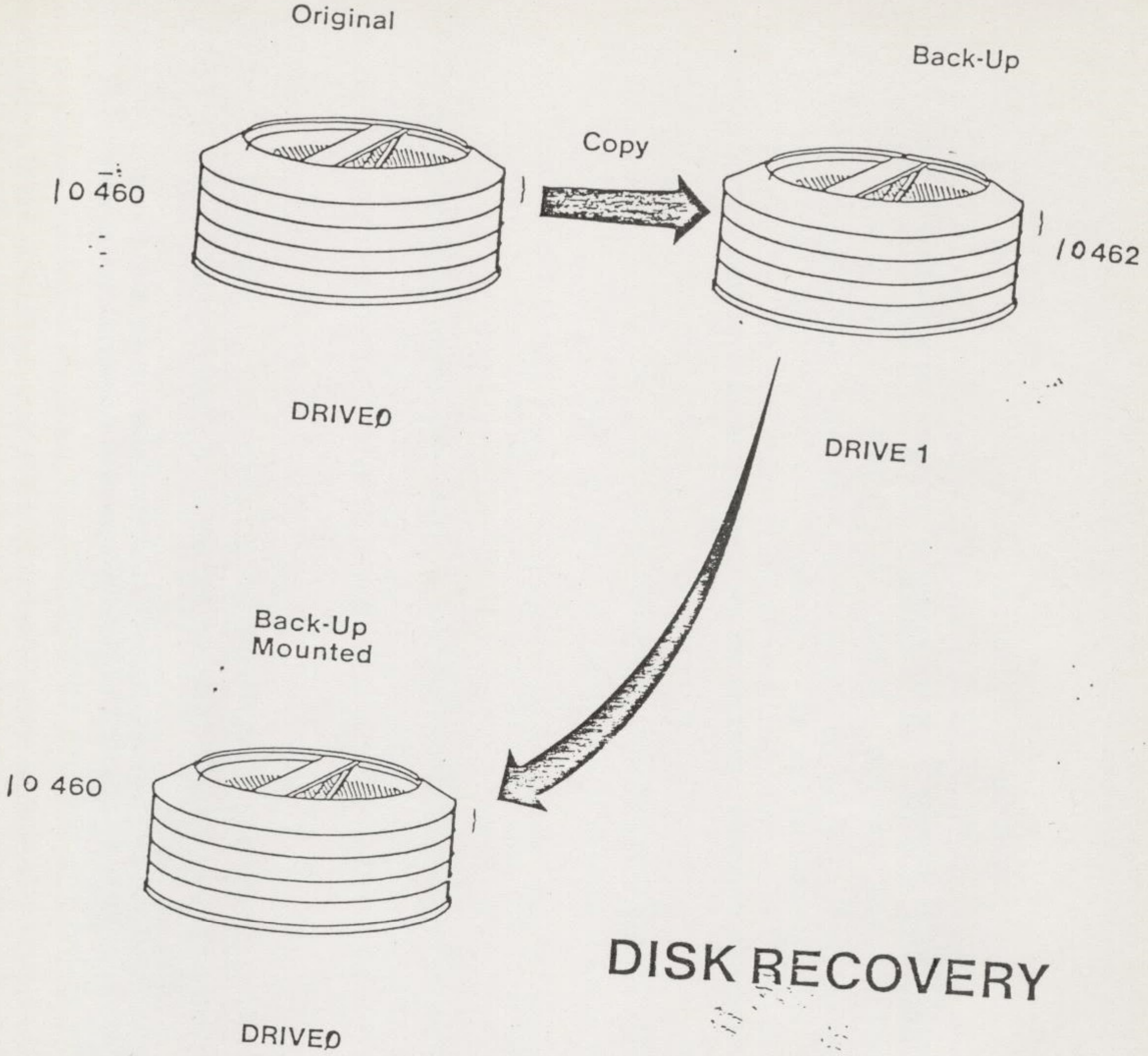
```
DISK NOT 462
ADD 462
```

- Job done!

BACKUPS



DISK TO DISK



DISK RECOVERY

COPY EXAMPLE

* IN ORDER TO BACKUP A PARTITION, YOU MUST SHUT DOWN THE PARTITION YOU WISH TO COPY FROM. SINCE YOU SHOULD BE MOUNTING A BACKUP DISK PACK, THE PARTITION YOU ARE COPYING TO IS ALREADY SHUT DOWN. THE FOLLOWING COMMANDS MUST BE GIVEN FROM THE SYSTEM CONSOLE.

```
SHUTDN 10460
DISK 10460 10462
```

* THE FOLLOWING IS THE TERMINAL SESSION FOR COPY

```
OK, AS DISK 10460
OK, AS DISK 10462
OK, COPY
COPY 16.4
FROM PHYS DISK= 10460
40MB STORAGE MOD? NO
TO PHYS DISK= 10462
40MB STORAGE MOD? NO
FROM, TO, RECORDS=  10460,  10462,   7407
PARAMETERS OK? YES

DONE
```

IF YOU ARE BACKING UP THE PARTITION THAT CONTAINS CMDNC0, YOU MUST DO SO UNDER PRIMOS II. THEN YOU DO NOT HAVE TO SHUT DOWN THE PARTITION OR ADD IT TO THE DISK ASSIGNABILITY TABLE.

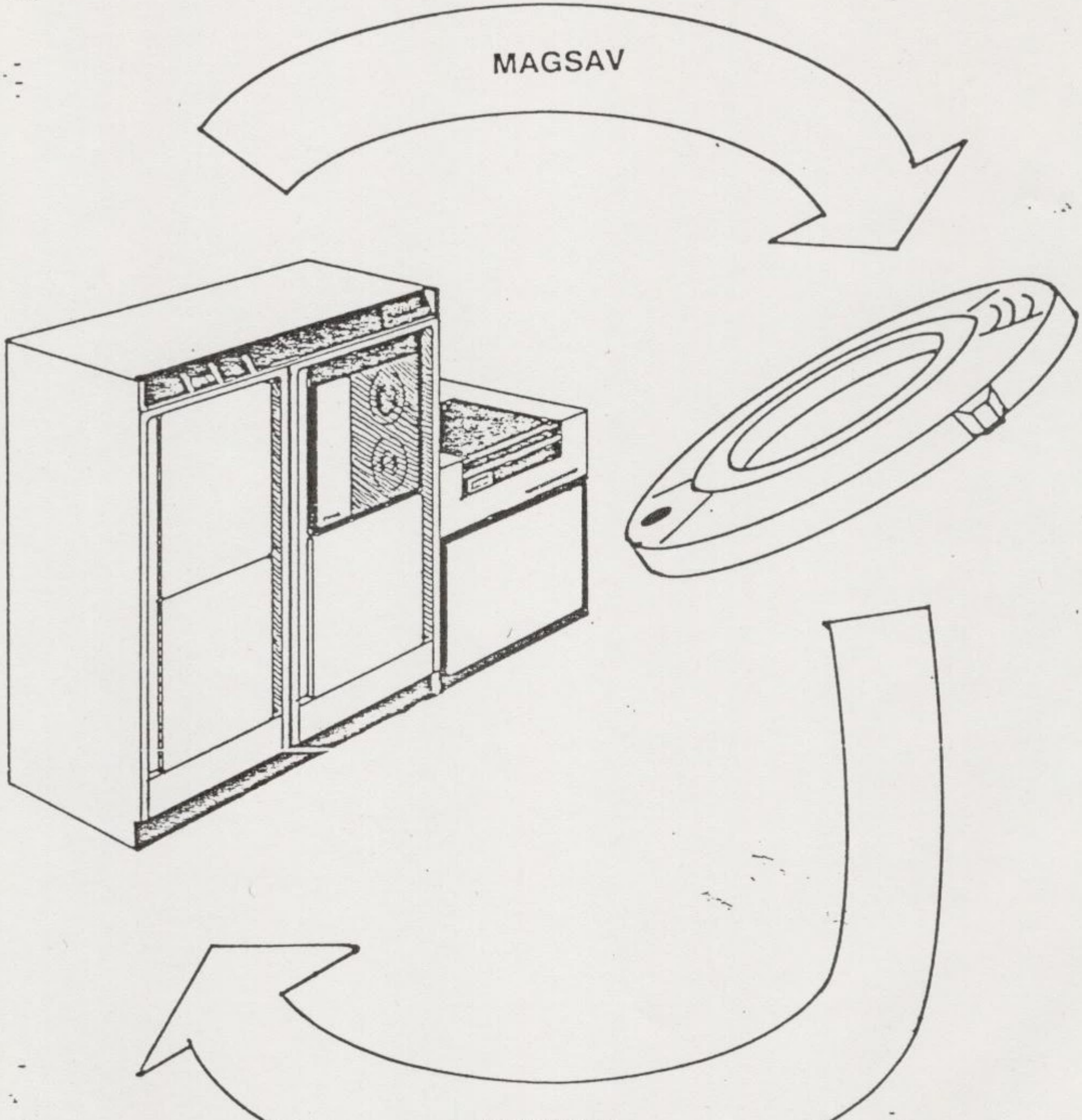
RECOVERING FROM DISK

* LILLIAN'S DIRECTORY WAS DELETED BY MISTAKE SO YOU MUST GET A COPY OF THE DIRECTORY OFF THE BACKUP DISK. FIRST, YOU MUST MOUNT THE BACKUP DISK ON THE SECOND DRIVE. THEN FROM A TERMINAL USE FUTIL TO MOVE THE UFD LILLIAN.U OVER TO THE OTHER DRIVE.

```
OK, A MFD SECRET 1

OK, L

UFD=MFD     1    OWNER

MASTER  MFD      BOOT     CMDNC0  DOS      NANCY.P  JACK1.P  GEORGE.U

OK, STAT DISK

DISK     LDEV  PDEV   SYSN
STUDNT     0     460
MASTER     1   10460
BACKUP     2   10462

OK, FUTIL
> TO MFD SECRET 1
> FROM MFD XXXXXX 2
> TRECPY LILLIAN
> Q

OK, L

UFD=MFD      1   OWNER

MASTER  MFD      BOOT     CMDNC0  DOS      NANCY.P  JACK1.P  LILLIAN.U
GEORGE.U
```

MAGNETIC TAPE UTILITIES

MAGNETIC TAPE UTILITIES



SETTING THE DUMP SWITCH

```
OK, A MFD SECRET 1

OK, L


UFD=MFD  1  OWNER


MASTER  MFD     BOOT    CMDNCO  DOS     NANCY.P  JACKI.P  LILLIAN.U
GEORGE.U


OK, AS MT1

OK, MAGSAV -L -UPDT
REV. 16.2
TAPE UNIT (9 TRK): 1
ENTER LOGICAL TAPE NUMBER: 1
TAPE NAME: BACKUP
DATE (MM DD YY):
REV NO:
NAME OR COMMAND: S1 B MFD1 6
NAME OR COMMAND: MFD
***START OF SAVE***

***END OF SAVE***
NAME OR COMMAND: SR
```

INCREMENTAL BACKUPS

```
OK, A MFD SECRET 1

OK, L

UFD=MFD  1  OWNER


MASTER  MFD     BOOT    CMDNCO  DOS     NANCY.P  JACKI.P  LILLIAN.U
GEORGE.U


OK, AS MT1

OK, MAGSAV -L -UPDT -INC
REV. 16.2
TAPE UNIT (9 TRK): 1
ENTER LOGICAL TAPE NUMBER: 1
TAPE NAME: BACKUP
DATE (MM DD YY):
REV NO:
NAME OR COMMAND: S1 B MFD1 6
NAME OR COMMAND: MFD
***START OF SAVE***

***END OF SAVE***
NAME OR COMMAND: SR
```

RESTORING A DIRECTORY

```
OK, A MFD SECRET 1
```

* ONE OF THE USERS, GEORGE TO BE EXACT, HAS ACCIDENTLY DELETED HIS WHOLE UFD. TO FIX THIS PROBLEM, YOU NEED TO MOUNT THE TAPE HIS UFD WAS SAVED ON. THE INDEX YOU RAN WHILE YOU WERE DOING THE SAVE WILL HELP YOU LOCATE THE PROPER TAPE.

```
OK, L

UFD=MFD  1  OWNER

MASTER  MFD     BOOT    CMDNCO  DOS     NANCY.P  JACKI.P  LILLIAN.U



OK, AS MT1

OK, MAGRST
REV. 16
TAPE UNIT (9 TRK): 1
ENTER LOGICAL TAPE NUMBER: 1
NAME: BACKUP
DATE(MM DD YY): 09-07-79
REV NO:       0
REEL NO:        1
READY TO RESTORE: S1 2
READY TO RESTORE: PARTIAL
TREENAME: MFD>GEORGE.U
TREENAME:

*** STARTING RESTORE ***

MFD > GEORGE.U
FILE COMPLETE

*** RESTORE COMPLETE ***


OK, L

UFD=MFD  1  OWNER

MASTER  MFD     BOOT    CMDNCO  DOS     NANCY.P  JACKI.P  LILLIAN.U
GEORGE.U
```

CONVENTIONAL TAPE BACKUP

```
OK, A MFD SECRET 1

OK, L


UFD=MFD  1  OWNER


MASTER  MFD     BOOT    CMDNC0  DOS      NANCY.P  JACKI.P  LILLIAN.U
GEORGE.U


OK, AS MT1

OK, MAGSAV -L
REV. 16.2
TAPE UNIT (9 TRK): 1
ENTER LOGICAL TAPE NUMBER: 1
TAPE NAME: BACKUP
DATE (MM DD YY):
REV NO:
NAME OR COMMAND: S1 B MFD1 6
NAME OR COMMAND: MFD
***START OF SAVE***

***END OF SAVE***
NAME OR COMMAND: SR
```

USAGE

USAGE

Provides system usage information as difference readings between successive invocations of the program

Runs as ring 3 process under standard operating system

Rev. 15 usage runs on the 64 user versions (with or without networking)

Counters may change whilst usage is looking at them so results can be inaccurate if time between runs is short - should no be less than.30 seconds

Segments 4, 6, 10 must be shared with read access from ring 3

To run:

```
1. R usage15  , followed by
   S          , at some time later
      further readings can be taken
      whenever "5" is typed

2. R usage15 1/n,
       Runs periodically, the time between
       Runs being n seconds (octal)
```

Outputs to terminal, use como to get results into a file

USAGE OUTPUT

LINE 1:

Date and time of run

DTME - Time between present and previous invocation in seconds

CPTOT - Total cpu time (seconds) used by all users since last cold start

IOTOT - Total I/O time (seconds) by all users since last cold start

Rest of output is difference between current and previous runs

LINE 2:

DCPTOT - Total CPU time (seconds) by all users

%CP - % of real time that CPU was running user processes (DCPTPT/DTIME)

DPFCN - Delta number of page faults

PF/SEC - Delta page faults per second (DPFCN/DTIME)

LINE 3:

DIOTOT - Total I/O time (paging and file)

%IO - % of real time that I/O was going on

DIOCN - Number of disk I/O requests (paging and file)

IO/SEC - Number of disk I/O requests per second (DIOCN/DTIME)

%OVLAP - Estimate of % of the I/O time that was overlapped with nonidle time (DIOCN - DCPBAK)/DIOCN)

LINE 4:

DLOCNT - Number of locate requests

LO/SEC - Locate requests per second (DLOCNT/DTIME)

DLOFCT - Number of locate hits on unused buffers

DLOSCT - Number of locate hits on same buffer

DLOUCT - Number of locate hits on used buffers

LINE 5:

DLOCCT - Number of locate misses

LM/SEC - Number of misses per second (DLOCCT/DTIME)

%MISS - % of locate requests which were misses (DLOCCT/DLOCNT*100)

%XCP - Unaccounted CP time (100 - the sum of %CPU) (Process exchange time)

LINE 6, SYSTEM PROCESSESS

%CLK - CPU for clock process

%AML - CPU for AMLC process

%MPC - CPU for MPC process

%IPC - CPU for IPC process

%FAR - CPU for farnet process

%SLC - CPU for SMLC process

%BAK - CPU for backstop process

USER DATA

USR - User number

LOGNAM - Login name

CUFD - Current UFD

MEM - Snapshot of number of pages in physical memory

CPTIME - Total CPU time since login (seconds)

DIF - CP - Delta CPU time (IE: IN DTIME)

%CP - Delta % CPU time ((DIF - CP)/DTIME)

IOTIME - Total I/O time since login (seconds)

DIF -IO - Delta I/O time (IE: IN DTIME)

%IO - Delta % I/O time ((DIF - IO)/DTIME)

Users only appear if their CP or I/O counters have changed since the last usage run

When a user logs in or out, will get incorrect data for that user on the next usage run

- Occasionally get negative numbers when counters overflow

```
OK, USAGE -FREQ 2
GO

 02/18/80 13:55:49.37  DTIME=   14.54  CPTOT= 4456.32  IOTOT= 2622.37

DCPTOT=   5.203  %CP=  35.793  DPFCN=    237  PF/SEC=  16.303
DIOTOT=  12.764  %IO=  87.802  DIOCN=    365  IO/SEC=  25.109  %OVLAP=  42.814
DLOCNT=  1176 LO/SEC=  80.90 DLOFCT=  1092 DLOSCT=     10 DLOUCT=      0
DLOCCT=     74 LM/SEC=   5.09 %MISS=   6.29   %XCP= 5.80
%CLK= 8.31 %AML= 4.73 %MPC= 0.00 %IPC= 0.00 %FAR= 0.00 %SLC= 0.15  %BAK=50.21

USR LOGNAM     MEM   CPTIME      DCP      %CP      IOTIME     DIO     %IO

  1  SYSTEM    153  1669.912   0.138    0.951    450.785   0.139   0.959
  3  NANCY       7     2.862   0.452    3.106      3.948   0.712   4.899
  6  SLUFD      12    33.986   1.008    6.931     31.621   3.988  27.433
  7  JACK1      27    13.241   3.394   23.344      8.791   3.258  22.409
  8  SLUFD      27     5.603   0.041    0.282     10.236   1.924  13.237
 19  SYSTEM      1   525.321   0.052    0.359     94.327   0.412   2.835
 20  FAM        16   100.637   0.066    0.451    186.176   0.855   5.878
 21  SYSTEM      1     6.105   0.053    0.366      5.821   0.327   2.251
```

STARTUP AND SHUTDOWN

SHUTDOWN

BEFORE YOU SHUTDOWN THE SYSTEM, YOU SHOULD WARN EVERYONE ON THE SYSTEM THAT YOU ARE SHUTTING DOWN. TO DO THIS, SEND A MESSAGE. BELOW IS AN EXAMPLE OF HOW TO SHUTDOWN THE SYSTEM. THIS PROCESS MUST BE DONE FROM THE SYSTEM CONSOLE.

```
OK, M ALL NOW
EVERYONE LOGOUT - THE SYSTEM IS GOING DOWN
```

AFTER EVERYONE HAS LOGGED OUT, LOGOUT THE PHANTOMS. IT MAY TAKE MORE THAN ONE MESSAGE TO GET EVERYONE OFF THE SYSTEM.

```
OK, LO ALL
```

ALL THE LOGOUT MESSAGES WILL NOW TYPE OUT ON THE SYSTEM CONSOLE.

```
OK, SHUTDN ALL
REALLY? YES
WAIT,
LOGICAL DEVICE 0, YOUR FILES ARE CLOSED
PRIMOS NOT IN OPERATION
```

SYSTEM STARTUP

TURN ON THE POWER IN THIS ORDER

CPU (TURN THE KEY TO ON)
DISK DRIVES (ONE AT A TIME)
TAPE DRIVES
OTHER PERIPHERAL DEVICES

BOOTING THE SYSTEM

TURN THE ROTARY SWITCH ON THE CPU TO STOP/STEP
PRESS MASTER CLEAR SWITCH
CHECK ADDRESS/DATA SWITCH SET TO ADDRESS
PRESS SENSE SWITCHES 10, 13, 14 UP (13 AND 14 IF USING CARTRIDGE DRIVE)
TURN ROTARY SWITCH TO LOAD
PRESS START SWITCH

AT THE SYSTEM CONSOLE

IF THE BOOT WAS SUCCESSFUL THE SYSTEM WILL PRINT -

```
      PHYSICAL DEVICE=
```

ON THE SYSTEM CONSOLE. YOU RESPOND WITH THE PHYSICAL DEVICE NUMBER OF YOUR
COMMAND SURFACE I.E. WHERE PRIMOS IS STORED.

TYPE WHAT IS UNDERLINED AT THE SYSTEM CONSOLE

```
        PHYSICAL DEVICE=460

        PRIMOS II REV 16 .....

        OK: STARTUP 460
        OK: A PRIRUN OR A PRI400
        OK: R PRIMOS
```

STARTUP FOR THE 50's SERIES

TURNING ON THE COMPUTER

TURN ON POWER TO THE CPU
TURN ON THE DISK DRIVES ONE AT A TIME
TURN ON THE POWER TO THE TERMINET
TURN ON THE REST OF THE PERIPHERAL DEVICES

BOOTING THE SYSTEM

THE SYSTEM CONSOLE WILL HAVE THE 'CP>' PROMPT
YOU TYPE IN :

```
CP> SYSCLR
CP> BOOT 114
```

THE SYSTEM CONSOLE WILL THEN PRINT OUT :

```
PHYSICAL DEVICE=
```

THE REST IS THE SAME AS THE 400 AND 500

** Listing of CONFIG File **

```
ABBREV YES
PAGDEV 20061       /* PAGING DEVICE IS
ALTDEV 20063       /* ALTERNATE PAGING DEVICE
ASRATE 1010        /* SYSTEM CONSOLE'S BAUD RATE IS 300
COMDEV 1060        /* COMMAND DEVICE
LOUTQM 144         /* INACTIVITY-LOGOUT TIME IS 100 MIN.
MAXPAG 2000        /* SPECIFY NUMBER OF PAGES OF MEMORY TO VALIDATE
NET ON             /* NETWORKS ARE TO BE CONFIGURED
SMLC ON            /* ENABLE SMLC LINES
NPUSR 5            /* SPECIFY NUMBER OF PHANTOM USERS
NRUSR 4            /* SPECIFY NUMBER OF REMOTE USERS
NTUSR 24           /* SPECIFY NUMBER OF TERMINAL USERS   .
NUSEG 65           /* SET NUMBER OF USER SEGMENTS PER USERS (default is 32)
TYPOUT YES
WIRMEM             /* PRINT SIZE OF WIRED MEMORY
AMLBUF 20 1500 1500
                   /* SET AMLC LINE 20'S INPUT & OUTPUT BUFFER SIZE IN WORDS
LOGLOG YES         /* ALLOW LOGIN WHILE LOGGED IN
ERASE              /* SET SYSTEM'S ERASE CHARACTER IF OTHER THAN " IS DESIRED
DISLOG YES         /* PERFORM LOGOUT WHEN AN AMLC LINE IS DISCONNECTED
GO
```

SYSTEMS HALTS

PRIMOS SYSTEM CRASH REPORT 400-500

WHEN THE SYSTEM HALTS DO THE FOLLOWING:

1. <u>DO NOT MASTER CLEAR AT THIS TIME.</u>

2. TURN ROTARY SWITCH TO <u>STOP/STEP.</u>

3. PUT ADDRESS/DATA TOGGLE SWITCH TO <u>ADDRESS</u> POSITION.

4. WRITE DOWN THE NUMBERS OF THE RED LIGHTS THAT ARE ON OR NOTE THE OCTAL VALUE. LIGHTS ON__________

5. NOW TURN ROTARY SWITCH TO <u>FETCH Y</u>.

6. PLACE ALL NUMBERED TOGGLE SWITCHES TO NEUTRAL POSITION (THE MAJORITY ALREADY ARE).

7. PRESS AND RELEASE <u>DATA CLEAR</u>.

8. PRESS AND RELEASE <u>START</u>.

9. PUT ADDRESS/DATA TOGGLE SWITCH TO <u>DATA</u> POSITION.

10. WRITE DOWN THE RED LIGHTS NOW ON. ADDRESS 0__________

11. TURN ROTARY SWITCH TO <u>FETCH Y + 1</u>.

12. PRESS AND RELEASE <u>START</u>.

13. WRITE DOWN THE RED LIGHTS NOW ON. ADDRESS 1__________

14. PRESS AND RELEASE <u>START</u>.

15. WRITE DOWN THE RED LIGHTS NOW ON. ADDRESS 2__________

16. TURN ROTARY SWITCH BACK TO <u>FETCH Y</u>.

17. PUT ADDRESS/DATA TOGGLE SWITCH TO <u>ADDRESS</u> POSITION.

18. PRESS AND RELEASE <u>DATA CLEAR</u>.

19. RAISE SWITCHES 1, 2, & 4.

20. PRESS NUMBERED SWITCHES 12, 13, 14, & 16 DOWN (THIS WILL TURN ON THEIR ASSOCIATED LIGHTS).

21. PRESS AND RELEASE <u>START</u>.

22. PUT ADDRESS/DATA TOGGLE SWITCH TO DATA POSITION.

23. WRITE DOWN THE RED LIGHTS NOW ON. 35 HI (DSWSTAT)__________

24. PUT SWITCH 4 IN NEUTRAL POSITION.

25. PRESS AND RELEASE START.

26. WRITE DOWN THE RED LIGHTS NOW ON. 35 LOW (DSWSTAT)__________

27. RAISE SWITCH 4.

28. PUT ADDRESS/DATA TOGGLE SWITCH TO ADDRESS POSITION.

29. PRESS AND RELEASE DATA CLEAR.

30. DEPRESS NUMBERED SWITCHES 12, 13, & 14.

31. PRESS AND RELEASE START.

32. PUT ADDRESS/DATA TOGGLE SWITCH TO DATA POSITION.

33. WRITE DOWN THE RED LIGHTS NOW ON. 34 HI (DSWRMA)__________

34. PUT SWITCH 4 IN NEUTRAL POSITION.

35. PRESS AND RELEASE START.

36. WRITE DOWN THE RED LIGHTS NOW ON. 34 LOW (DSWRMA)__________

37. PUT ADDRESS/DATA TOGGLE SWITCH TO ADDRESS POSITION.

38. PRESS AND RELEASE DATA CLEAR.

39. RAISE SWITCH 4, DEPRESS SWITCHES 12, 13, 14 & 15.

40. PRESS AND RELEASE START.

41. PUT ADDRESS/DATA TOGGLE SWITCH TO DATA POSITION.

42. WRITE DOWN THE RED LIGHTS NOW ON. 36 HI (DSWPB)__________

43. PUT SWITCH 4 IN NEUTRAL POSITION.

44. PRESS AND RELEASE START.

45. WRITE DOWN THE RED LIGHTS NOW ON. 36 LOW (DSWPB)__________

46. NOW DO A WARM START. IF YOU CAN'T DO A WARM START YOU HAVE TO DO A COLD START. TO DO A WARM START, TURN THE ROTARY SWITCH TO STOP/STEP, PRESS MASTER CLEAR SWITCH, THEN PRESS THE START SWITCH TWICE. '*** WARM START ***' SHOULD PRINT OUT ON THE SYSTEM CONSOLE IF A WARM START IS POSSIBLE. ALL THE TERMINALS SHOULD BEGIN TO FUNCTION. IF THE WARM START IS NOT SUCCESSFUL, YOU SHOULD GO THROUGH COLD START PROCEDURES. THESE ARE THE SAME AS A NORMAL STARTUP.

SYSTEM HALTS ON A 50'S SERIES

WHEN YOUR SYSTEM HALTS, THE LIGHT ABOVE THE MASTER CLEAR BUTTON GOES ON AND THE TERMINALS STOP WORKING. THE SYSTEM CONSOLE SHOULD HAVE PRINTED THE HALT LOCATION. RECORD THESE NUMBERS IN YOUR SYSTEM LOG. BELOW IS A PROCEDURE FOR FINDING THE REASON FOR THE HALT.

```
CP> D DSWSTAT
CP> D DSWRMA
CP> D DSWPB
```

RECORD THE NUMBERS THAT PRINT ON THE SYSTEM CONSOLE IN RESPONSE TO THESE COMMANDS. NOW YOU ARE READY TO ATTEMPT A WARM START. TYPE THE FOLLOWING COMMANDS ON THE SYSTEM CONSOLE.

```
CP> SYSCLR
CP> RUN

HALTED AT :   1001:  000010

CP> RUN

*** WARM START ***
```

IF YOU ARE SUCCESSFUL WITH THE WARM START ATTEMPT, ALL THE USERS WILL BE ABLE TO CONTINUE. IF THE WARM START ATTEMPT WAS NOT SUCCESSFUL, YOU MUST THEN TRY A COLD START. THIS IS THE SAME PROCESS AS NORMAL SYSTEM STARTUP.

THE EVENT RECORDER

```
OK,LOGPRT TTY
LOGPRT REV 16.3
INPUT TREENAME: CMDNCO>LOGREC


 ***** CMDNCO>LOGREC, 22:20:12 FRI 25 JAN 1980 *****

 09:25:00 FRI 18 JAN 1980
 ________________________________

 MEMORY PARITY (ECCC) DSWSTAT= 020110 146400 DSWRMA= 000006 017253
     DSWPB= 000006 017367 PPN,WN= 000024 001253 BIT=  6 OP=1

 09:25:52 FRI 18 JAN 1980
 ________________________________

 SHUTDOWN BY OPERATOR

 09:27:20 FRI 18 JAN 1980
 ________________________________

 COLD START  CPU TYPE= 6  MICROCODE REV= 2
     ID= 000000 000006 000000 000002 000000 000000 000000 000000

 DISK MOUNT: OP/SYS ON 000460

 09:27:36 FRI 18 JAN 1980
 ________________________________

 DISK MOUNT: ANLYS1 ON 010460

 DISK MOUNT: MRKREP ON 020460

 DISK MOUNT: ADMIN  ON 030460

 DISK MOUNT: CUST1  ON 041060

 DISK MOUNT: CUST2  ON 061060

 DISK MOUNT: SCRTCH ON 110061

 09:28:04 FRI 18 JAN 1980
 ________________________________

 DISK MOUNT: SFTWAR ON 000462

 DISK MOUNT: ANLYS2 ON 010462

 DISK MOUNT: DEMOPK ON 020063
```

NATIONAL INSERTABLE-TAB INDEXES ENABLE YOU TO
MAKE YOUR OWN SUBJECT ARRANGEMENT, USING PLAIN
INSERTS ON WHICH TO WRITE YOUR OWN CAPTIONS.

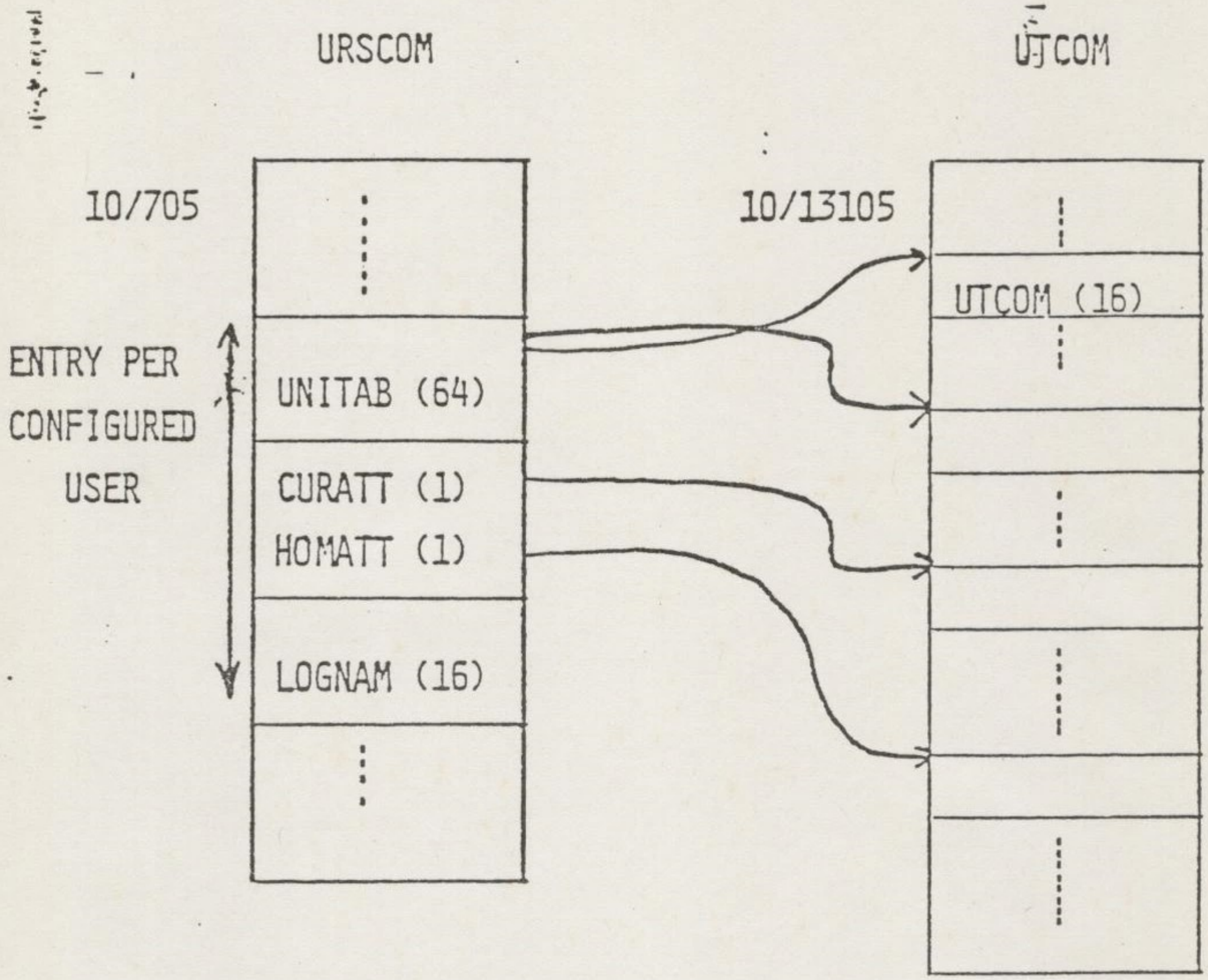
The Beaded edge on tab makes it easy to insert captions

Made in U. S. A.

NATIONAL

23-680

NEW STRUCTURE OF USRCOM AND UTCOM:



ONLY 6 CHARACTERS OF LOGNAM ARE USED AT REV. 16. UNITAB, CURATT AND HOMATT ARE 16 BIT POINTERS INTO UTCOM. WHEN USED AS AN ATTACH POINT, UTCOM DOES NOT KEEP ASCII CHARACTER STRING (USE UFDNAM.)

ATA STRUCTURES OF THE PRIMOS FILE SYSTEM DISK

1.1 OVERVIEW

The format of a Primos disk is similar for all disk types supported by Prime. Each logical disk consists of a series of sequentially numbered records. Each disk record consists of a record header and a data section. All records of a given logical disk are the same size; every record has a record header. Disk records are used to contain all data on the disk including directories. Primos currently supports two record sizes. Storage Modules have 1040 word records divided into a 16 word record header and a 1024 word data section. All other Prime supported disks have 448 word records divided into an 8 word record header and a 440 word data section.

1.2 RECORD HEADER FORMATS

1.2.1 Overview

The data items in the record header of both Storage Modules and all other disks are the same. The size of each data item and the order of the items in the record header are different.

Below will be discussed the meaning of each data item and its usage. The name of the data item is the name used to referenct the item in Primos IV operating system FORTRAN code.

REKCRA Current Record Address
The record address (record number) of this record will generally be checked by the disk driver (DVDISK).

REKPOP Beginning Record Address or Father Record Address
For all records except the first record in a SAM string, this data item contains the record address of the first record in the file (BRA). If the record is the first record in the file, REKPOP contains the beginning record address of the directory in which the file is entered. If the file is a DAM file and the record is the first record in an index level, but not the highest index level, REKPOP contans the record address of the first record in the next highest index level (SAM string).

REKDCT Record Data Count
Number of words which are valid in the data section of the record. If the record is not the last record in a SAM string, the data count must be the maximum allowed for the record.

REKTYP File Type
The item is only valid in the first record of each file (BRA). In all other records, REKTYP must be zero.

```
Bit 16:    Ø => SAM file, 1 => DAM file
    15: -- 1 => segment directory, else Ø
    14:    1 =>.UFD, else Ø
Bits 2-13: on record Ø (BRA of BOOT) and record 2 (BRA of DSKRAT)
           only, 1 if disk has 1Ø48 word records (Storage Module);
           else Ø.
```

REKFPT Forward Pointer
Record address of next record in SAM string. Zero if current record is last record in SAM string.

REKBPT Back Pointer
Record address of previous record in SAM string. Zero if current record is first record in SAM string.

REKLVL Index Level
Zero if a SAM file. Else, In a DAM File the index level of the SAM string of wich the current record is a member. The highest index level has the numerically highest numer; the data level is zero.

1.2.2 Record Header Format - 1Ø48 word records (Storage Module)

Ø	REKCRA	INTEGER*4
2	REKPOP	INTEGER*4
4	REKDCT	INTEGER*2
5	REKTYP	INTEGER*2
6	REKFPT	INTEGER*4
8	REKBPT	INTEGER*4
1Ø	REKLVL	INTEGER*2
11	reserved	5 INTEGER*2 words, must be zero
15		

1.2.3 Record Header Format - 448 word records

Ø	REKCRA	INTEGER*2
1	REKBRA	INTEGER*2
2	REKFPT	INTEGER*2
3	REKBPT	INTEGER*2
4	REKDCT	INTEGER*2
5	REKTYP	INTEGER*2
6	REKLVL	INTEGER*2
7	reserved	INTEGER*2, must be zero

1.2.4 Accessing Record Beader Data Items

The ring Ø subroutine LOCATE is used to access both the record header and the data section of a disk record. Details on the usage of LOCATE are given elsewhere in this document.

One of the actions of LOCATE is to arrange the record headers so that the data item lengths are those given for 1Ø48 word records. The proper method of accessing the variables from FORTRAN code is:

```
        I = REKCRA (BUFNEW)
```

and similarly. Note that each data item must be accessed individually; note ordering of the data items can be assumed.

1.3 STRUCTURE OF FILES

1.3.1 Overview

All collections of information on a Primos file system disk are organized into files. Directories are files whose data sections contain "special" information. Two basic types of files are currently supported, SAM (Sequential Access Method) and DAM (Direct Access Method). There is no difference in the user interface to access information in either SAM or DAM files. Thus the editor will work on either type of file without any special coding conventions.

1.3.1.1 SAM Files

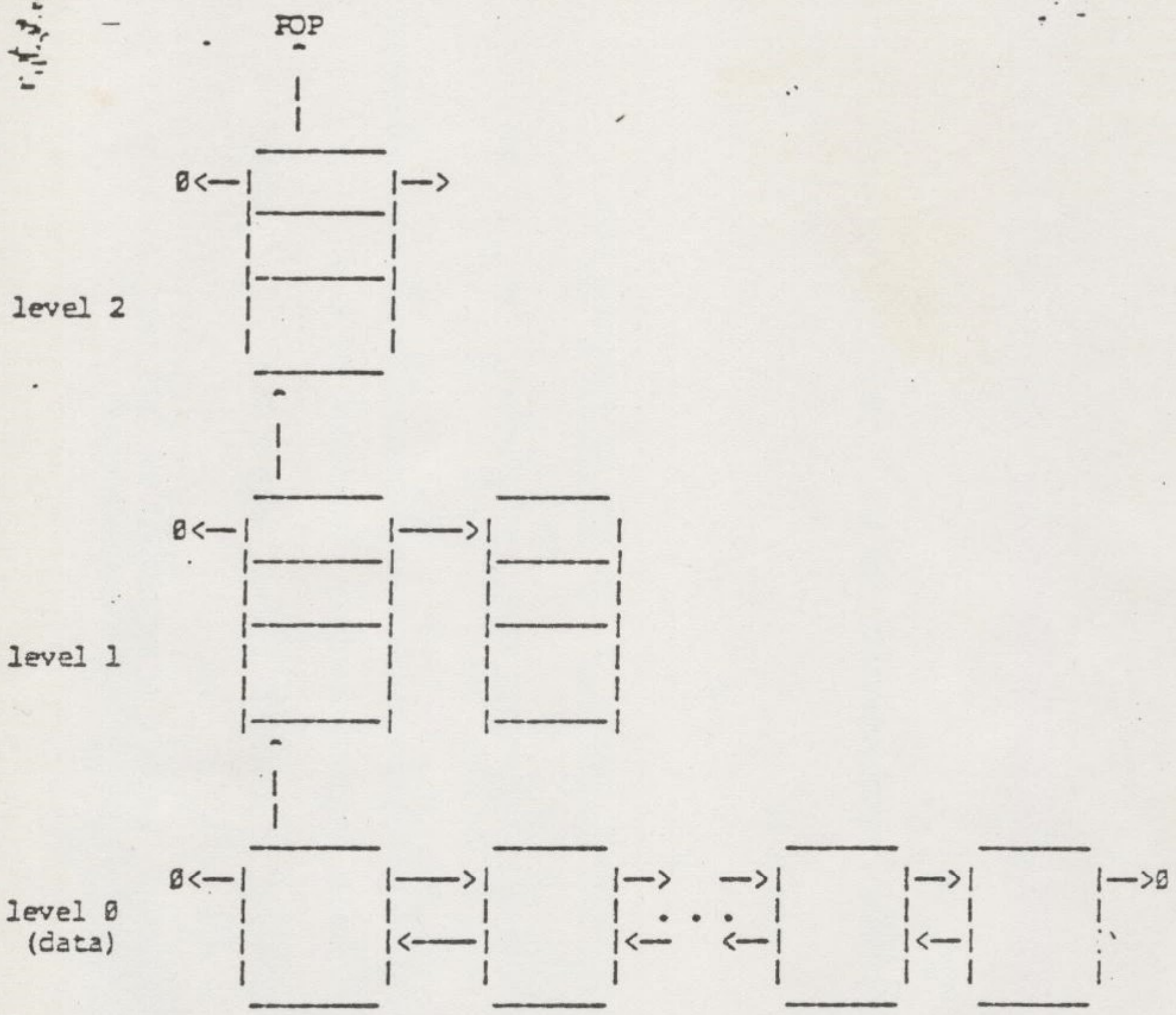
A SAM file consists of a single "SAM string" in which all the records in the file are linked together in a linear doulby linked list using the pointer REKFPT and REKBPT in the record headers of the records in the file.

```
          FOP
           ^
           |
           |
        ______        ______      ______
   Ø<--|      |----->|      |-->|      |-->Ø
       |      |      |      |   |      |
       |      |<-----|      |<--|      |
       |      |      |      |   |      |
        ______        ______      ______
```

Ø<— The data in any SAM file may be accessed using PRWF$$ either sequentially or random access. Random accesses wich are relatively far apart will be slower than if the file were a DAM file.

1.3.1.2 DAM Files

A DAM file consists of a hierarchy of "SAM strings". The data in a DAM file may be accessed either randomly or sequentialy using PRWF$$. Either type of access will occur with approximately the same speed.



Pictured above is a moderate size (514 data records on a Storage Module) DAM file. Note that each index level including the data level is a SAM string. That is the records in each level are linked together in a linear doubly linked list using REKFPT and REKBPT in the record headers. REKPOP in the record header of the first record in each level points to its "father", either the first record in the immediately superior level or the BRA of the directory in which the file is entered. The data words of all records which are not in the data level contain pointers to (record addresses of) records in the immediately inferior level. The top level index is constrained to be exactly one record long.

1.3.2 EXTENDING AND TRUNCATING DAM FILES

When a DAM file is newly created it consists of two records. The beginning record address (BRA) is that of the index record. The index record will have a data count of 2 (record addresses always INTEGER*4 even on 448 word disks) as the data section will contain one pointer pointing to the data record of the file. As user data is written to the file, records will be chained into the data level and record address pointers added to the index record until the data section of the index record is full (512 data records on Storage Modules, 22Ø data records on all other disks). Since the top level index is constrained to be one record long, another level of index must be created in order to grow the file. The next level of index is created by logically adding another record to the existing index and then creating another higher level index which contains 2 two record address pointers, each of which points to the two lower level index records. This is done by the ring Ø procedure NEWDAM and the COPYUP entry to the ring Ø module LOCATE in such a manner that the BRA (physical record address) is still the first record in the file (logically the newly created higher level index) while the data that was formerly in the physical BRA is copied to a freshly acquired record.

When a DAM file is truncated, the number of index levels is never reduced. The number of records in each SAM string can be truncated to one. Thus, if the DAM file pictured above is truncated to zero data words, the structure will be changed to:

```
                   FOP
                 ________
           Ø<—|           |—>Ø
              |-----------|
level 2       |           |
              |           |
                 ________
                 |
                 |
                 ________
           Ø<—|           |—>Ø
              |-----------|
level 1       |           |
              |           |
                 ________
                 |
                 |
                 ________
           Ø<—|           |—>Ø
level Ø       |           |
 (data)       |           |
                 ________
```

1.3.3 STRUCTURE OF DIRECTORY FILES

1.3.3.1 Overview

There are two types of directories currently supported by Primos: (1) User File Directories (UFDs) and (2) Segment Directories (SEGDIRs). Note that a directory is itself a file and may be either a SAM or DAM file. Currently, DAM UFDs are not supported. The structure of record header and index record pointers as outlined above is valid for all directories. The directory "information" is entirely in the data section (of the data level, if DAM) of the records wich make up the directory file.

UFDs are always accessed in a sequential manner, usually looking for a match on file name. File entries in a UFD allow for flexible setting of attributes such as protection, date and time modified, etc.

SEGDIRs may be accessed either randomly or sequentially. File entries in a SEGDIR consist of only the beginning record address of the inferior file; all attributes are derived from the UFD entry of the topmost SEGDIR in a hierarchy of SEGDIRs. Only data files and other SEGDIRs can be entered (inferior to) a SEGDIR. That is a UFD is not allowed under a SEGDIR.

1.3.3.2 UFD Structure

1.3.3.2.1 Overview

All UFds are SAM files. All information within a UFD is contained in "UFD entries". Each entry starts with an Entry Control Word (ECW). The left byte of the ECW (bits 1-8) contans the UFD entry type and the right byte (bits 9-16) contain the length of the entry in 16 bit words. Each UFD entry type has a fixed length header (which may be zero length) and zero or more sub-entries. Each sub-entry has a Sub-entry Control Word (SWC) containing sub-entry type and length similar to the ECW. Thus, the internal format of a UFD is somewhat self-defining. In order to allow forward and backward compatibility, all code which deals with VFD entries is written so that "unknown" entry and sub-entry types are ignored. The length field is used to skip over unknown types.

Currently there are 3 defined UFD entry types.

1	UFD header
2	Vacant entry
3	File entry

1.3.3.2.2 UFD Header

The UFD header is always the first entry in every UFD. It contains the owner and non-owner passwords.

0	1 24	
1		Owner password (3 words)
4		Non-owner passwords (3 words)
7		Reserved, must be 0 16 words
23		

1.3.3.2.3 File Entry

The file entry is used to enter a file (data or directory) in a UFD. The entry contans the internal name (BRA), external name (character string), and attributes.

L - Filename length in wds

Ø	3|12+L	ECW (Entry Control Word)
1	BRA	Beginning Record Address
3	Reserved	Must be zero 3 words
6	PROTEC	Protection
7	Reserved	Must be zero
8	DATMOD	Date last modified
9	TIMMOD	Time last modified
1Ø	FILTYP	Tile type
11	Ø|L +1	SCW (Subentry Control Word)
12	FILNAM	File Name
n		

PROTEC Bits 1-8 Owner Rights
Bits 9-16 Non-Owner Rights in each byte
1 read
2 write
4 truncate/delete

DATMOD Bits: 1-7 Year
8-11 Month
12-16 Day

TIMMOD (Seconds since Midnight)/4

FILTYP Bits 5-6 reader/writer concurrency lock
0 => system default
1 => reader xor 1 writer
2 => n readers xor 1 writer
3 => n readers AND n writers

Bit 4: 1 if "special" file (BOOT,DSKRAT,
(MFD,BADSPT)
Bits 9-16: file type
0 => sam data
1 => dam data
2 => Sam SEGDIR
3 => dam SEGDIR
4 => UFD

FILNAM File name is a left justified, blank padded character string (ASCII). The filename may be 1 to 32 characters (1-16 words) in length. Thus, the length field in the SCW ("l") must be between 2 and 17.

1.3.3.2.4 Vacant Entry

The vacant entry type is used to logically delete a file entry. The contents of all words in the entry other than the ECW are undefined. Space compression is not done so that existeng file entries do not change relative position within the UFD. The "get position " and "set position" functions of RDEN$$ require the file entries not move.

1.3.3.3 SEGMENT DIRECTORY STRUCTURES

1.3.3.3.1 Overview

SEGDIRs contain only internal names (BRA) or null entries (INTL(0)).

1.3.3.3.2 Structure

```
      _______
  Ø  | BRA Ø |    Beginning Record Address
     |       |        (file in entry Ø
      -------
  2  | BRA 1 |
     |       |
      -------
  4  |   Ø   |    Null Entry
     |   Ø   |    (no file in entry 2)
      -------
     | . . . |
     |       |
      -------
  2n | BRA n |
     |       |
      -------
```

1.3.3.4 SPECIAL FILES

1.3.3.4.1 MFD

The MFD (Master File Directory) is the root node of the hierarchial file structure. The MFD is a UFD. The BRA of the MFD is defined to be 1. There is a file entry for "MFD" in the MD. One of the passwords of the MFD must be "xxxxxx".

1.3.3.4.2 Disk Record Availability Table

The "DSKRAT" is a sam data file entered in the MFD which contains a bit-map which indicates which records on the disk belong to files and which are free. The name of the logical disk is the character string name given to the dskrat file. The BRA of the dskrat is defined to be 2.

Ø	8	length of dskrat header
1	RECSIZ	number of words in disk record (inc. header)
2	NRECS	number of records in partition (INTEGER*4)
4	NHEADS	number of heads in prtition
5	Reserved .	Must be zero
8	BITMAP	Map of used/free records 1 bit/record

Users should never change the data in the DSKRAT file. Typically (i.e., when "xxxxxx" is the MFD non-owner password) the protection should be set to 1 1 (read only rights for both owner and non-owner).

1.3.3.4.3 BOOT

The sam data file BOOT is the record zero bootstrap used to read in and start the PRIMOS II operating system. The BRA of BOOT is defined to be Ø.

1.3.3.4.4 BADSPT

The sam data file named BADSPT is entered in the MFD by the disk formatting utility MAKE. It contains the heads and track numbers of disk records which are known to be unreadable. The file is only used by the disk consisting verification utility FIXRAT.

1.3.3.4.5 INTERNAL DATA BASES

1.3.3.4.5.1 N1LOCKS

N-readers-one-writer locks, or "n1locks", allow concurrent use and interlocked updating of a database. An n1lock may be locked for "writing" (exclusive use or update) or for reading (non-exclusive use).

The file system uses a collection of <u>ordered</u> n1locks. They are ordered in the sense that they must be locked only in priority order [i.e., a process cannot lock a priority 1 lock while holding a priority 4 lock). This prevents the classic deadlock situation in which process 1 has locked A and needs B [where priority (B) > priority (A)] while process 2 hqas locked B and needs A [process 2 would be in priority violation].

The six file system locks are described following.

FSLOK [File System Global Lock]

- Held for reading whenever referencing ANY file system databas. Prevents addition or shutdown of disks.
- Held for writing during addisk, shutdown-disk, and certain special cases of SRCH$$ (change-access).

UFDLOK [UFD Lock]

- Held for reading whenever any directory is being searched.
- Held for writing whenever any directory will be (or could be) modified (e.g., creating a file).

UTLOK [Unit Table Lock]

- Held for writing whenever referencing the Unit Table, to prevent changes to that table by other processes. In particular, the Open operation conflict check is interlocked in this way.

TRNLOK [Transaction Lock]

- Used to ensure that a given read or write call will never be interleaved with another read or write on the same shared file. Held for reading or writing as appropriate. Some operations on segment directories use this lock.

RATLOK [Record Available Table Lock]

- Held for writing whenever the RAT for a given disk is being accessed. Serializes disk allocation and deallocation.

DSKLOK [Disk DIM Lock]

- Used to single-thread the Disk DIM. Always held for writing.

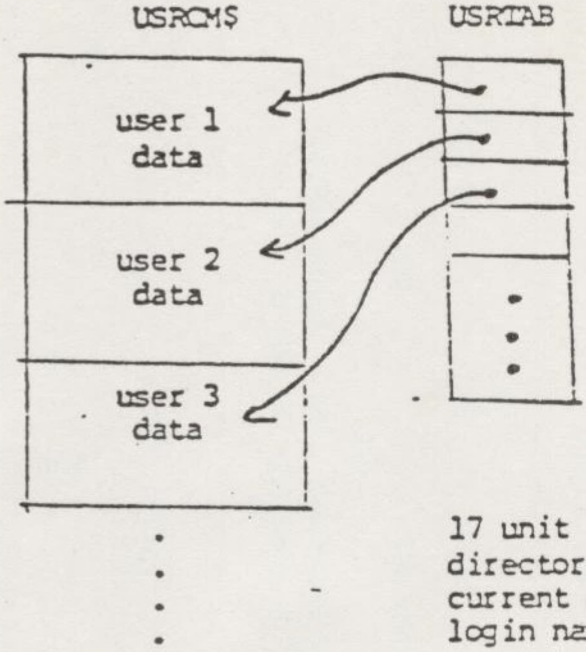
LOCSEM [Locate Semaphore, not an nllock]

- Used for mutual exclusion in critical regions of the LOCATE routine.

Note that, for most nllocks, recursive locking is not allowed (e.g., Process cannot lock A if it already has A locked). The only exception is FSLOK, which may be recursively locked for reading, or locked for reading after being locked for writing, but not locked for writing after being locked for reading.

1.3.3.4.5.2 UNIT TABLE DATABASE

1.3.3.4.5.2.1 Layout of Usrcm



17 unit table entries home
directory information
current directory information
login name

1.4.5.2.2 UNIT TABLE ENTRY

ord	name	contents
Ø	vstat	bit 1: modified; bits 2-8: filetype (4=dir, 2=segdir, 1=dam); bits 9-16: open access (1=read, 2=write, 3=RW). If file closed: all Ø.
1	vbra	(2 words) Beginning Record Address of File.
3	vdvno	Logical disk of file.
4	vdcra	(2 words) Current position Rec Addr (of Dam Index), dam files only. -1 if invalid.
6	vdrwp	(2 words) Ordinal position, in records.
8	vcra	(2 words) Current position Rec Addr of data record.
1Ø	vrwp	Ordinal position, offset in record indicated by vdrwp.
11	vpriv	bits 1-8: access control setting of file; bits 9-16: per-file RW Lock.
	vpopra	points to date/time modified field in parent directory entry. (2 words) Rec. Addr.
14	vpoprw	word offset for vpopra.

4.5.2.3 HOME/CURRENT DIRECTORY INFORMATION

wc	contents
Ø	(16 words) entryname of directory.
16	(2 wordds) Beginning Record Addr of directory.
18	logical disk of directory
19	Record Addr of parent of directory
21	bits 1-8: Ø=nonowner, 1=owner bits 9-16: access control information.
22	length of entryname.
23	Record Addr of DTM in parent directory entry.
25	offset in record of DTM

3.4.5.3 File System Internal Subroutines

1.3.3.4.5.3.1 Close file by unit or name

close (bra,dvno,unit,code)

bra,dvno	point to file if unit=Ø, else ignored.
unit	is specific unit if >Ø, or Ø if close (bra,dvno)
code	standard error code (Output)

restriction: cannot go remote.

1.3.3.4.5.3.2 Change Open Access

cngacc (key,unit,type,code)

key	1(read), 2(write), 3(RW)
unit	unit wose acc is to be changed. Must be open
type	file type of <unit>. (Output)
code	standard error code (Output)

restrictions: no remote. New access must not conflict with other users. Unit table st not be locked on call.

1.3.3.4.5.3.3 Delete a Directory Entry

delete (dvno,bra,aldpr,enthed,entpos,code)

dvno	logical disk of file
bra	beginning rec addr of file
oldpar	true if an old part'n
enthed	first word of file's directory entry
entpos	(int*4) position of enthed in the parent directory
code	standard error code. (Output)

estrictions: TRNLOK must not be locked on call. UFDLOK should be locked for writing round call.

.3.3.4.5.3.4 Delete All Records in a File

elrec (bra,dvno,filpop,code)

bra	Beginning Rec Addr of file to be gutted.
dvno	logical disk of file
filpop	B.R.A. of parent directory of file
code	standard error code. (Output)

st tion: RATLOK must not be locked on call.

3.3.4.5.3.5 Search Directory for Named File

```
ufd (name,length,dirpos,dirent,code)
```

```
    name          name of file to be looked up.
    length        bits 5-10:  directory select
                  (0 = user cufd, :77 = susr curd,
                  other = that logical disk mfd)
    dirpos        points to start of directory entry, or
                  suitable hole if file not found (Output)
    dirent(29)    dirent(1) = 1 if old part'n, 0 if new
                  dirent (2:29) = copy of directory entry if
                  file found, else dirent(2) = size of hole in
                  words for new prt'n only (Output)
    code          standard error code (Output)
```

restrictions: UFDLOK must be set for reading (at least). TRNLOK must not be locked at call. UNIT D will be used to open source directory. It will be left open and positioned to DIM slot for a file found.

1.3.3.4.5.3.6 Allocate a Disk Record

```
newrec = getrec (ra,dvno,code)
```

```
    ra        record address of current place in file.  New rec
              will be allocated "near" this one if possible.
    dvno      logical disk on which to allocate.
    code      Standard error code.  (Output)
    newrec    record addr of new record, if allocated.
```

restrictions: RATLOK must not be locked at call. Must not be called for remote disk.

1.3.3.4.5.3.7 Compare Two File System Entrynames

```
equal = nameq$ (name1,length1,name2,length2)
```

```
    name1      is first name.
    length1    is length (name1) in characters.
    name2      is second name
    length2    is length (name2) in characters.
    equal      is true if names are equal.  (Output)
```

Note: Lower case is converted to upper case.

1.3.3.4.5.3.8 Add Record to New-Partition DAM File

```
newdam (drwp,dvno,datsiz,nrall,cra,bra,dcra,code)
```

triction: must not go remote.

.3.3.4.5:3.9 Create a New Entry in Current Directory

nwbra = newfil (name,length,pos,dirent,type,code)

name	name of entry to be created
length	length of name in chars.
pos	position in directory of hole in which to write new entry (int*4)
dirent(29)	directory entry in same format as fsufd. (Output)
type	type of file to create.
code	Standard error code. (Output)
newbra	B.R.A. of new file. (Output)

restrictions: cannot go remote. UFDLOK must be held for write. RATLOK cannot be locked at call.

1.3.3.4.5.3.1Ø Allocate Space on Disk for New File

newbra = newfll (oldpar,type,dvno,filpop,code)

oldpar	true if an old partition.
type	type of file being created.
dvno	logical disk on which to create
filpop	BRA of parent directory.
code	standard error code. (Output)
newbra	BRA of new file's space. (Output)
unitx	index in unit table of unit NOT to be checked in this scan. Ignored if -1.
fildev	logical disk of file in questin.
fbra	BRA of file in question
rwlock	desired RWl lock settime to check [Ø = exclusive, 1 = n readers x or 1 writer, 3 = n readers of 2 writer, 5 = open]
fop	desired open mode (1 = R, 2 = W, 3 = RW, 4 = Delete, CName, etc.)
OK	true if no conflict. (Output)

restrictions: must be called with UTLOK held at least for reading.

1.3.3.4.5.3.11 Perform SRCH$$ Functions on Segment Dir

bra = schseg (key,segunt,unit,type,code)

All arguments from corresponding args to SRCH$$.

restrictions: may not go remote. UFDLCK and UTLOK and TRNLOK must not be set at call.

3.3.4.5.3.12 Check If File System Entryname Legal

xto$ (name, length,trulen,CK)

```
name     is the name to check
length   length (name) in chars.
trulen   length (name) less trailing blanks.  (Output)
CK       -true if name is OK
```

1.3.3.4.5.3.13 Truncate File to Current Position

trunc$ (unit,code)

```
unit   is file unit to be truncated.
code   is standard error code.  (Output)
```

restrictions: may not go remote. UTLOK must not be locked at call. TRNLOK must not be locked at call.

1.3.3.4.5.3.14 Add or Shut Down Disk

trwrat (key,ldev0)

```
key    1 = add, 2 = shud down
ldev   logical disk to do.
```

restrictions: must be called with FSLOK held for writing.

1.3.3.4.5.3.15 Associative Buffer Manager

locate (key,ra,ldev)

```
key    bit 1:  bypass read if set
       bit 2:  demote previous buffer if set
       bit 16:  mark new buffer modified if set
ra     record addr to oprate on
ldev   logical disk of <ra>
```

restrictions: must not be called with DSKLOK set.

2 SUPPORT FOR NEW DEVICES

2.1 1600/6250 Tape Drive Support

At Revision 16, PRIMOS IV has been modified to include software support for 1600/6250 BPI tape drives. For complete details, see Section 3 and Section 4.

3.3 User QUIT Handling - QUIT$

A new (and *temporary*) direct entrance call is provided in Revision 16 PRIMOS IV that will allow a user program running in ring 3 to determine if a QUIT has taken place. This call is designed to be used only when QUITs have been inhibited by a call to BREAK$.

Example:

```
CALL QUIT$ (LOGICAL)
IF (LOGICAL) GO TO handle_quit
```

This call will return .TRUE. only if QUITs are inhibited and the user has attempted to QUIT. If a QUIT was pending (i.e., .TRUE. is returned), the pending QUIT is cleared and will not take place when BREAK$ is called to reenabled QUITs. Calls to QUIT$ will never reset user terminal input and output buffers. A separate direct entrance call is provided for that purpose.

The QUIT$ call is a *temporary* facility in PRIMOS IV and is *subject to change or removal in the future*. QUIT$ is not available in the FORTRAN library.

3.4 Clearing User Terminal Buffers - TTY$RS

A new (and *temporary*) direct entrance call is provided in Revision 16 PRIMOS IV to allow a process to clear its own terminal input and output buffers. This facility is useful in certain cases (e.g. when a process elects to handle its own QUITs).

Example:

```
CALL TTY$RS (KEY, CODE)
```

KEY is an INTEGER*2 variable which specifies which buffers are to be cleared. A value of :100000 specifies the output buffer, :40000, the input buffer, and :140000, both buffers. CODE is an INTEGER*2 variable that will contain an error code upon return from TTY$RS.

TTY$RS can be called when a user ring program decides that input to the program that has already been typed is to be discarded. This might be useful, for example, in a case where a text editor detects an error in its input and wishes to ignore further input that the user has already typed.

The TTY$RS call is a *temporary* facility in PRIMOS IV and is *subject to change or removal in the future*. TTY$RS is not available in the FORTRAN library.

3.5 CPU and LOGIN Time Limits - LIMITS

A new direct entrance call is provided in Revision 16 PRIMOS IV to allow a process to lower its CPU and/or LOGIN time limits.

Name: LIMITS

Purpose:

The subroutine LIMTS is called to alter or read the amount of cpu or login time a process (user) is limited to. Each process (user) possesses a cpu and login time limit which are initially defined to be infinite.

The maximum finite value either of these limits may be set to is 1000000 (decimal). The login time limit is measured in minutes, and the cpu time limit is measured in seconds. If either of these limits is ever exceeded, the process (user) is logged out.

Usage:

```
CALL LIMITS (key + subkey, LIMIT, RESERV, CODE)
```

key

is the operation to be performed on the limit. Valid operations are K$READ (1, read current limit value), and K$WRIT (2, set limit value).

subkey

is the target limit that "key" operates on. Valid target limits are K$CPLM (:400, CPU time limit) and K$LGLM (:1000, LOGIN time limit).

LIMIT

is an INTEGER*4 variable which receives the value of the target limit when "key" is K$READ, and which contains the value for the target limit when the "key" is K$WRIT.

RESERV

is an INTEGER*2 variable which is reserved for future use. The value of RESERV must be 0.

CODE

is an INTEGER*2 variable that (upon return from a call to LIMITS) is set to 0 if no error has occurred. If the call to LIMITS was unsucessful, CODE may be set to E$BKEY or E$BPAR. E$BKEY is returned if the "key + subkey" is an invalid combination (see NOTES). E$BPAR is returned if LIMIT is either negative or greater than the current limit, or RESERV is nonzero.

Notes: The following describes the only valid "key+subkey" combinations:

KSREAD + KSCPLM returns in LIMIT the remaining cpu time until forced logout occurs in _seconds_. A value of zero means that the limit is infinite.

KSREAD + KSLGLM returns in LIMIT the remaining login time until forced logout occurs in _minutes_. A value of zero means that the limit is infinite.

KSWRIT + KSCPLM sets the cpu time until forced logout to LIMIT _seconds_ from now. The cpu time until forced logout may not be raised.

KSWRIT + KSLGLM sets the login time until forced logout to LIMIT _minutes_ from now. The login time limit until forced logout may not be raised.

Example:

```
     CALL LIMIT$(KSWRIT+:400, 0000010, RESERV, CODE)
```

In this example, the CPU time limit is set to 10 seconds.

The LIMITS call is a _temporary_ facility in PRIMOS IV and is _subject to change or removal in the future_. LIMITS is not available in the FORTRAN library.

3.6 TSMT -- New Instructions

The following instructions have been added to TSMT. (TSMT is described in the Reference Guide, Software Library.) These instructions are only valid with version two and three magnetic tape controllers. Use of these instructions with older versions of the controller will cause an error message to be printed and the command to be aborted.

Octal	Hex	Action
100020	8010	Erase a 3 inch gap on the tape.
100040	8020	Unload. Completely rewind the tape and place the drive offline.
100100	8040	Set density to 1600 BPI (PE)
100120	8050	Set density to 6250 BPI (GCR)
040500	0940	Read record backwards.

3.6.1 Erase 3 Inch Gap

This operation causes a 3 inch gap to be erased from the tape. This is useful in error recovery schemes.

3.6.2 Unload

This operation causes the tape be completely rewound, and the drive to be placed offline. This is useful in preventing accidental use of the tape drive before the tape has been removed from the drive.

3.6.3 Density Selection

It is assumed that tapes are written with one density. This assumption is enforced by only permitting changes in density at the load point. For this reason, it is not necessary, or possible, to set the density when reading a tape. When the first record is read, the density of the tape is determined. The rest of the tape will be read (or written) using that density.

For example, if the user set the density to 6250 BPI with the ASSIGN command and read the first record of a 1600 BPI tape, then the rest of the tape would be read using 1600 BPI. If after reading that record, a record was written onto the tape (without rewinding to the load point); then that record would also be written at 1600 BPI. If the tape was rewound and then a record was written, the density would be switched to 6250 BPI. Although the density setting of 6250 BPI is remembered, it will not go into affect until a record is written at the load point.

If the user assigns a tape without specifying a density, the unit will left at the density from the previous use. The default density (at system initialization time) is 1600 BPI.

3.6.4 Read Record Backwards

This request causes the tape to read a record while moving the tape backwards. It is sometimes possible to read a record backwards when a bad tape prevents reading the record in the forward direction. After the record is read, it will be necessary to reorganize the data. The words of the record will be in reverse order. Each word will have the bytes reversed. The bits within each byte will be in correct order.

3.12 UPS - Uninterruptible Power Supply Support

PRIMOS IV now supports an Uninterruptible Power Supply. If a power failure should occur, and a site has UPS support, power to the backplane is maintained via batteries. When normal power is restored, an automatic warm-start will be performed after a slight delay (to allow the disk(s) to build up to the proper number of RPMs). The delay is set by the CONFIG directive UPS. A power-fail entry is written to the LOGREC file by LOGPRT when power is restored. See the 'UPS' CONFIG directive in Section 7 for more details.

4.3 ASSIGN Command Modification

The ASSIGN command has been extended to allow the setting of the density for 1600/6250 tape drives which use the version three magnetic tape controller (MPC-3).

```
ASSIGN MTn [WAIT] [-6250BPI] [-1600BPI]
```

-6250BPI Set the density to 6250 BPI. The default is 1600 BPI for a software settable drive. This control argument is only valid for the 1600/6250 BPI tape drive.

-1600BPI Set the density to 1600 BPI. This control argument is only valid for a software settable drive.

4.4 CHAP Command Modification

A user may now lower the priority of his own process by specifying the LOWER control argument.

```
CHAP LOWER n
```

This command will lower the priority of the user's process by "n" levels. The value of "n" must be 0 <= n <= 7. If n = 0, the priority of the process is unchanged; otherwise, the process' priority is lowered by 'n' levels. If the resultant level is less than the lowest, then the priority of the process is set to the lowest. The LOWER control argument can only be used from a user process, not from the system console (process 1).

4.5 LOGOUT Command Modification

The LOGOUT command has been modified so that when 'LOGOUT ALL' is specified from the system console (user 1) the remote file access manager (FAM) is not logged out if it is a running process.

4.6 LOOK Command Modification

The LOOK command has been modified so that a 'REALLY?' prompt is issued for any LOOK command whose request is considered to be risky or dangerous to system integrity. (If the LOOK command involves an attempt to do a FROM from a segment that does not exist, an attempt to do a TO to a segment that does exist, or attempts to map either shared or stack segments with write permission, the command is considered risky or dangerous to system integrity.) A simple 'YES' will allow the operation to proceed.

PERMIT and DENY affect only disk partitions already started up at the time of the REMOTE command. Disks shut down and started up again will get the system default permissions until an explicit REMOTE PERMIT or REMOTE DENY command changes them. The system default permissions are determined from the file NETCON which is created by NETCFG. The REMOTE PERMIT command will not automatically add a disk to any system. The REMOTE DENY command will not revoke a system's existing access to a disk.

4.10 STARTUP Command Modification

The STARTUP command has been extended to permit a disk to be software write-protected.

A disk is write-protected by specifying PROTECT in the STARTUP command as follows:

```
     STARTUP PROTECT dvno1 [dvno2 ... dvno5]
```

PROTECT may only be specified for disks which are started locally, and does not govern the rights of remotely added disks. Remotely added disks assume the write-protection status of the local system.

The status of the write-protect feature may be changed for a given partition by respecifying the STARTUP or ADDISK command with or without PROTECT.

If an subsequent STARTUP command is issued for the same disk, and PROTECT is not specified, the write-protect feature is disabled. (An STARTUP PROTECT to an already protected disk does not change the protection.) If an STARTUP PROTECT command is issued for a disk which does not have protection enabled, it is important that the disk be shutdown first, to insure that the disk is not inadvertently written upon.

4.11 UNASSIGN Command Modification

The UNASSIGN command has been extended to allow an unload operation for tape drives. This control argument is only valid for a version two controller (MPC-2) and a version three controller (MPC-3) which controls 1600/6250 BPI tape drives.

```
     UNASSIGN MTn [-UNLOAD]
```

-UNLOAD   Rewind the tape completely, and set the drive offline before unassigning the drive.

5 EXTERNAL COMMAND MODIFICATIONS AND ADDITIONS

5.1 MTDENS

MTDENS allows the user to set the density on a magnetic tape drive from the command level _under PRIMOS II_. The ASSIGN command performs this function under PRIMOS IV.

```
     MTDENS     MTn  [-6250BPI]  [-1600BPI]
```

MTn Magnetic tape drive identifier (MT0 - MT7).

-6250BPI Set the density to 6250 BPI. The default is 1600 BPI for a software settable drive. This control argument is only valid for the 1600/6250 BPI tape drive.

-1600BPI Set the density to 1600 BPI. This control argument is only valid for a software settable drive.

```
100   IF (STATV(1).EQ.0) GOTO 120 /* SEE IF IO IS ALREADY DONE
          CALL T$MT (UNIT,LOC(0),0,:100000,XSTATV) /* WAIT
          GOTO 100
120   . . .
```

10.2 Error Recovery for Tape Writes

There are many possible error recovery schemes. The two that are described here are based on different record formats. The first algorithm can be used when records contain only data. The other scheme requires that the records contain extra information for error recovery.

Note: The following schemes are provided as alternatives to using the IOCS routines that FTN uses. The error recovery provided in the IOCS routines correspond to that described for Simple Write Error Recovery.

10.2.1 Simple Write Error Recovery

The aim of the simple error recovery program is to get by a possible bad spot on the tape by erasing part of the tape where the error occurred and rewriting the record after that gap.

The program does not try to rewrite the record on the same spot on the tape even though repeated tries on the same spot may improve the tape enough to permit the write to succeed. The tape is considered marginal at that spot and may not be readable at a later date.

Only the version three controller (MPC-3), which supports the 6250 bpi tape drives, has an erase command. On other controllers, the tape can be erased by writing a file mark and then backspacing over the file mark. This will cause three inches of tape to be erased.