BASEMENT

PROPERTIES OF STORAGE HIERARCHY SYSTEMS WITH MULTIPLE PAGE SIZES AND REDUNDANT DATA

Chat-Yu Lam

Stuart E. Madnick

April 10, 1979

Revised

CISR #42

Sloan WP No. 1047--79

This material is based upon work supported, in part, by the National Science Foundation under Grant No. MCS77-2Ø829.

Center for Information Systems Research

Massachusetts Institute of Technology

Sloan School of Management

77 Massachusetts Avenue

Cambridge, Massachusetts, 02139

PROPERTIES OF STORAGE HIERARCHY SYSTEMS WITH MULTIPLE PAGE SIZES AND REDUNDANT DATA

Chat-Yu Lam

Stuart E. Madnick

April 10, 1979

Revised

CISR #42
Sloan WP No. 1047-79

This material is based upon work supported, in part, by the National Science Foundation under Grant No. MCS77-2Ø829.

Center for Information Systems Research

Massachusetts Institute of Technology
Alfred P. Sloan School of Management
50 Memorial Drive
Cambridge, Massachusetts, 02139
617 253-1000

PREFACE

The Center for Information Systems Research (CISR) is a research center of the M.I.T. Sloan School of Management; it consists of a group of Management Information Systems specialists, including faculty members, full-time research staff, and student research assistants. The Center's general research thrust is to devise better means of designing, generating and maintaining application software, information systems and decision support systems.

Within the context of the research effort sponsored by the National Science Foundation under Grant No. MCS77-20829, CISR proposes to investigate the architecture of the INFOPLEX Data Base Computer which is particularly designed for large-scale information management. INFOPLEX applies the theory of hierarchical decomposition in its design and makes use of multiple microprocessors in its implementation to obtain high performance, high reliability, and large storage capacity. Research issues to be addressed include optimal decomposition of information management functions into a functional hierarchy to be implemented by a hierarchy of microprocessors, and optimal physical decomposition of a data storage hierarchy to support the memory requirements of the information management functions.

In Technical Report No. 1, we discussed the INFOPLEX concept and its research directions. This report focuses on the study of a generalized data storage system for very large databases which can be used to support the memory requirements of INFOPLEX. This data storage system makes use of multiple page sizes in a hierarchy of storage levels and maintains multiple copies of the same information across the storage levels. Important properties of such a data storage system are derived here.

ABSTRACT

The need for high performance, highly reliable storage for very large on-line databases, coupled with rapid advances in storage device technology, has made the study of generalized storage hierarchies an important area of research.

This paper analyzes properties of a data storage hierarchy system specifically designed for handling very large on-line databases. To attain high performance and high reliability, the data storage hierarchy makes use of multiple page sizes in different storage levels and maintains multiple copies of the same information across the storage levels. Such a storage hierarchy system is currently being designed as part of the INFOPLEX database computer project. Previous studies of storage hierarchies have primarily focused on virtual memories for program storage and hierarchies with a single page size across all storage levels and/or a single copy of information in the hierarchy.

In the INFOPLEX design, extensions to the Least Recently Used (LRU) algorithm are used to manage the storage levels. The Read-Through technique is used to initially load a referenced page, of the appropriate size, into all storage levels above the one in which the page is found. Since each storage level is viewed as an extension of the immediate higher level, an overflow page from level 'i' is always placed in level 'i+1'. Important properties of these algorithms are derived. It is shown that, depending upon the types of algorithms used and the relative sizes of the storage levels, it is not always possible to guarantee that the contents of a given storage level 'i' is always a superset of the contents of its immediate higher storage level 'i-1'. The necessary and sufficient conditions for this property to hold are identified and proved. Furthermore, it is possible that increasing the size of intermediate storage levels may actually increase the number of references to lower storage levels, resulting in reduced performance. Conditions necessary to avoid such an anomaly are also identified and proved.

Key Words and Phrases : database computer, very large databases, data storage hierarchy, storage management algorithms, inclusion properties, modelling, performance and reliability analysis.

CR Categories : 4.3 Supervisory Systems,
4.33 Data Base
5.2 Metatheory
6.22 Special-Purpose Computers
6.34 Storage Units

TABLE OF CONTENTS

1. Introduction

Two and three-level memory hierarchies have been used in practical computer systems [5, 9, 13]. However, there is relatively little experience with general hierarchical storage systems. Rapid advances in storage technology coupled with the need for high performance, highly reliable on-line databases makes the idea of using a generalized storage hierarchy as the repository for very large shared data bases very attractive.

One major area of theoretic study of storage hierarchy systems in the past has been the optimal placement of information in a storage hierarchy system. Three approaches to this problem have been used: (1) Static placement [1, 4, 22] - this approach determines the optimal placement strategy statically, at the initiation of the system; (2) Dynamic placement [7, 16] - this approach attempts to optimally place information in the hierarchy, taking into account the dynamically changing nature of access to information; (3) Information structuring [11, 14] - this approach manipulates the internal structure of information so that information items that are frequently used together are placed adjacent to each other.

Another major area of theoretic study of storage hierarchy systems has been the study of storage management algorithms [2, 3, 8, 10, 17, 21]. Here the study of storage hierarchy and the study of virtual memory systems for program storage have overlapped considerably. This is largely due to the fact that most of the studies

of storage hierarchies in the past have been aimed at providing a virtual memory for program storage. These studies usually do not consider the effects of multiple page sizes across storage levels, nor the problem of providing redundant data across storage levels. These considerations are of great importance for a storage hierarchy designed specifically for very large data bases.

Madnick [15, 18, 19] proposed the design of a generalized storage hierarchy for large data bases that makes use of multiple data redundancy against failure and multiple page sizes in different storage levels for high performance. Such a storage hierarchy system is to be used in the INFOPLEX database computer [12, 20].

Conceptually, the INFOPLEX database computer consists of a functional hierarchy and a physical (storage) hierarchy (See Figure 1).

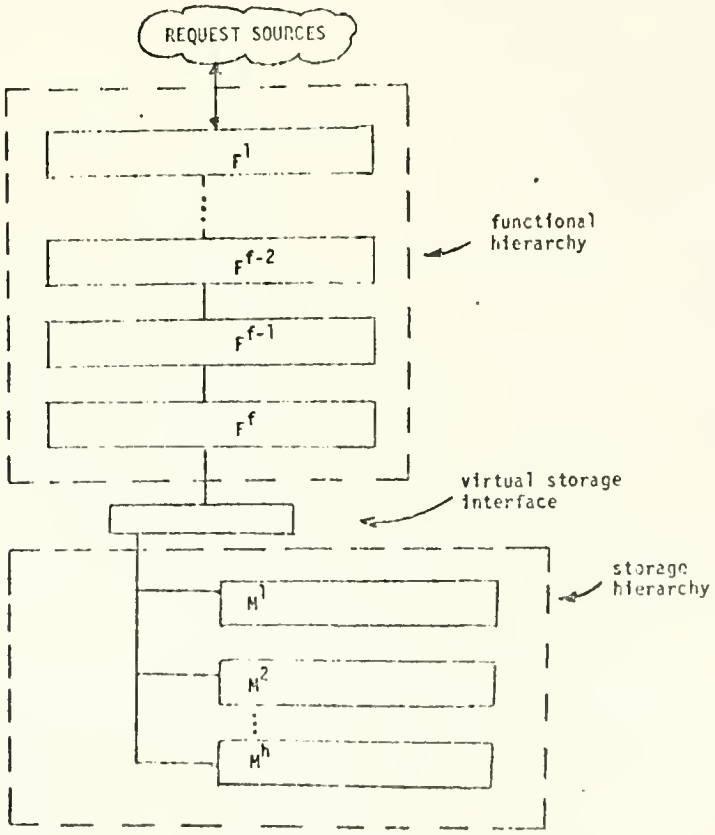

FIGURE 1 INFOPLEX Data Base Computer Conceptual Organization

The functional hierarchy implements all the information management functions of a database manager, such as query language interpretation, security verification, and data path accessing, etc. In INFOPLEX, the functional hierarchy is implemented using multiple microprocessors. Both pipeline and parallel processing are exploited to realize high performance and high reliability. To support the storage requirements of the functional hierarchy, INFOPLEX makes use of a generalized data storage hierarchy system.

In this paper, we extend this work by developing a model of the data storage hierarchy, proposing extensions to the Least Recently Used (LRU) algorithm for managing the storage hierarchy, and deriving important properties of the data storage hierarchy.

2. Model Of A Data Storage Hierarchy

A Data Storage Hierarchy consists of h levels of storage devices, M^1, M^2, ..., M^h. The page size of M^i is Q_i and the size of M^i is m_i pages each of size Q_i. Q_i is always an integral multiple of Q_{i-1}, for i=2,3 . . ., h. The unit of information transfer between M^i and M^{i+1} is a page, of size Q_i. Figure 2 illustrates this model of the Data Storage Hierarchy.

All references are directed to M^1. The storage management algorithms automatically transfer information among storage levels. As a result, the Data Storage Hierarchy appears to the reference source as a M^1 storage device with the size of M^h.

As a result of the storage management algorithms (to be discussed next), multiple copies of the same information may exist in different storage levels.

2.1. Storage Management Algorithms

We shall focus our attentions on the basic algorithms to support the read-through [18] operation. Algorithms to support other operations can be derived from these basic algorithms.

In a read-through, the highest storage level that contains the addressed information broadcasts the information to all upper storage levels, each of which simultaneously extracts the page (of the appropriate size) that contains the information from the broadcast. If the addressed information is found in the highest storage level, the read-through reduces to a simple reference to the addressed information in that level. Figure 3 illustrates the read-through operation.

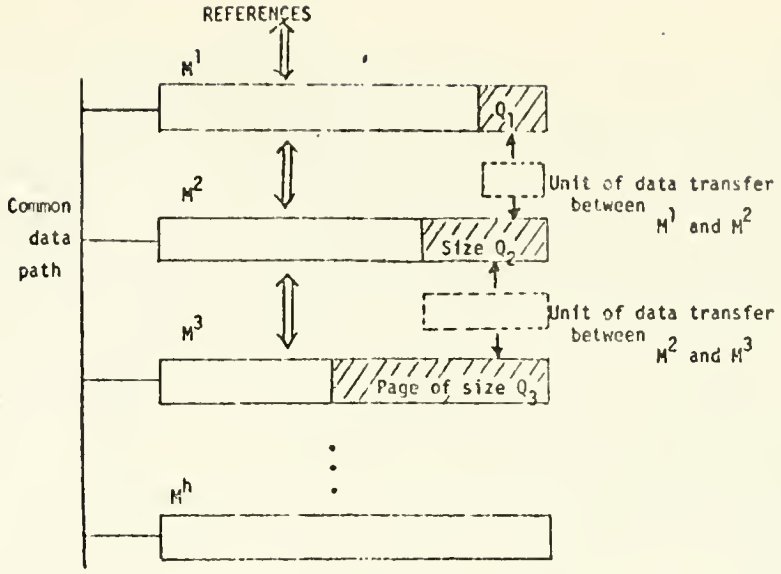

Figure 2 Model of a data storage hierarchy system

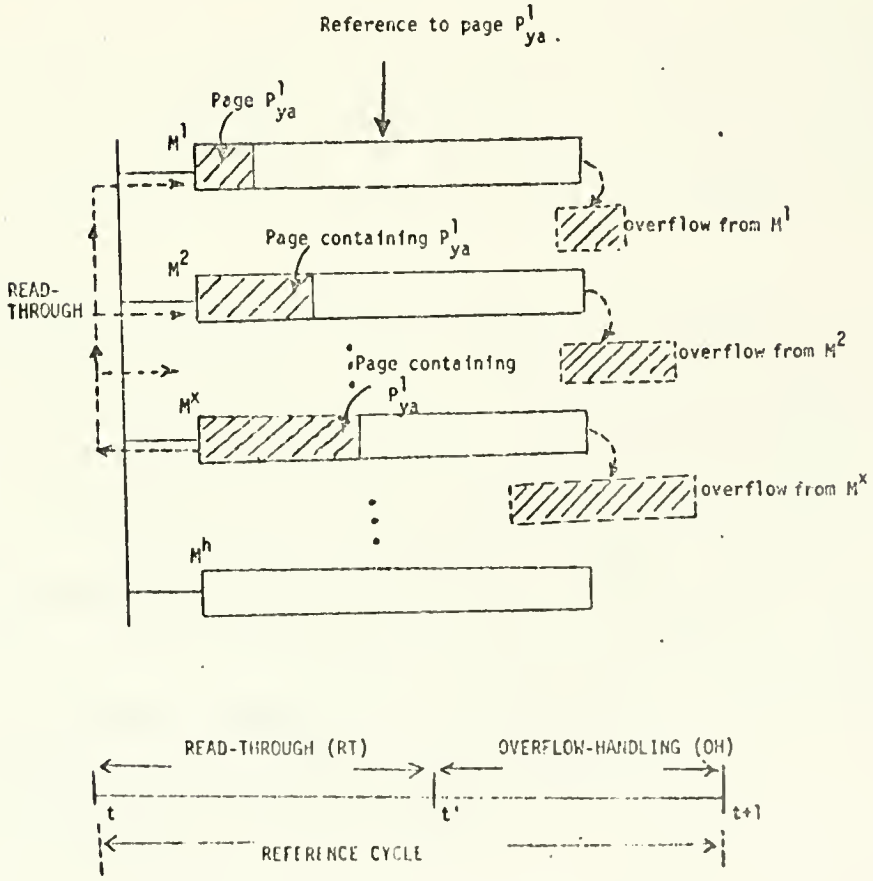

Figure 3 Illustration of the READ-THROUGH operation

Note that in order to load a new page into a storage level an existing page may have to be displaced from that storage level. We refer to this phenomonon as overflow. Hence, the basic reference cycle consists of two sub-cycles, the read-through cycle (RT), and the overflow handling cycle (OH), with RT preceeding OH.

For example, Figure 3 illustrates the basic reference cycle to handle a reference to the page P^1_{ya}. During the Read-Through (RT) subcycle, the highest storage level (M^x) that contains P^1_{ya} broadcasts the page containing P^1_{ya} to all upper storage levels, each of which extracts the page of appropriate size that contains P^1_{ya} from the broadcast. As result of the Read-Through, there may be overflow from the storage levels. These are handled in the Overflow-Handling (OH) subcycle.

It is necessary to consider overflow handling because it is desirable to have information overflowed from a storage level to be in the immediate lower storage level, which can then be viewed as an extension to the higher storage level.

One strategy of handling overflow to meet this objective is to treat overflows from M^i as references to M^{i+1}. We refer to algorithms that incorporate this strategy as having dynamic-overflow-placement (DOP).

Another possible overflow handling strategy is to treat an overflow from M^i as a reference to M^{i+1} only when the overflow information is not already in M^{i+1}. If the overflow information is already in M^{i+1}, no overflow handling is necessary. We refer to algorithms that incorporate this strategy as having static-overflow-placement (SOP).

Let us consider the algorithms at each storage level for selecting the page to be overflowed. Since the Least Recently Used (LRU) algorithm [6, 21] serves as the basis for most current algorithms, we shall consider natural extensions to LRU for managing the storage levels in the Data Storage Hierarchy system.

Consider the following two strategies for handling the Read-Through Cycle. First, let every storage level above and including the level containing the addressed information be updated according to the LRU strategy. Thus, all storage levels lower than the addressed information do not know about the reference. This class of algorithms is called LOCAL-LRU algorithm. This is illustrated in Figure 4.

The other class of algorithms that we shall consider is called GLOBAL-LRU algorithm. In this case, all storage levels are updated according to the LRU strategy whether or not that level actually participates in the read-through. This is illustrated in Figure 5.

Although the read-through operation leaves supersets of the page P^1_{ya} in all levels, the future handling of each of these pages depends upon the replacement algorithms used and the effects of the overflow handling. We would like to guarantee that the contents of each storage level, M^i, is always a superset of its immediately higher level, M^{i-1}. This property is called Multi-Level Inclusion (MLI). Conditions to guarantee MLI will be derived in a later section.

It is not difficult to demonstrate situations where handling overflows generates references which produce overflows, which generate yet more references. Hence another important question to resolve is to determine the conditions under which an overflow from M^i is always found to already exist in M^{i+1}, i.e., no reference to storage levels lower than M^{i+1} is generated as a result of the overflow. This property is called Multi-Level Overflow Inclusion (MLOI). Conditions to guarantee MLOI will be derived in a later section.

We shall consider these important properties in light of four basic algorithm alternatives based on local or global LRU and static or dynamic overflow. Formal definitions for these algorithms will be provided after the basic model of the Data Storage Hierarchy system is introduced.

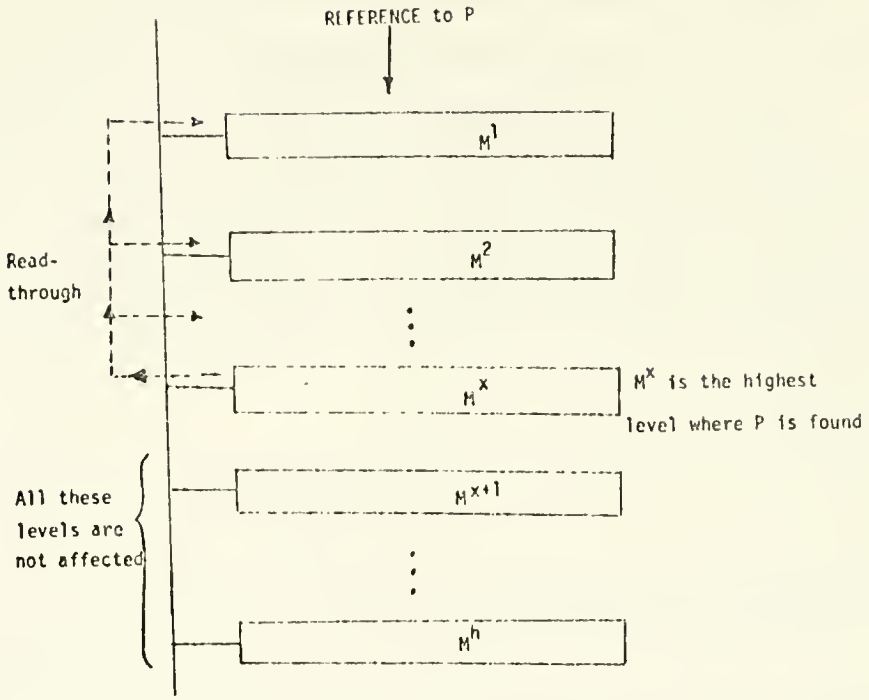

Figure 4 LOCAL-LRU

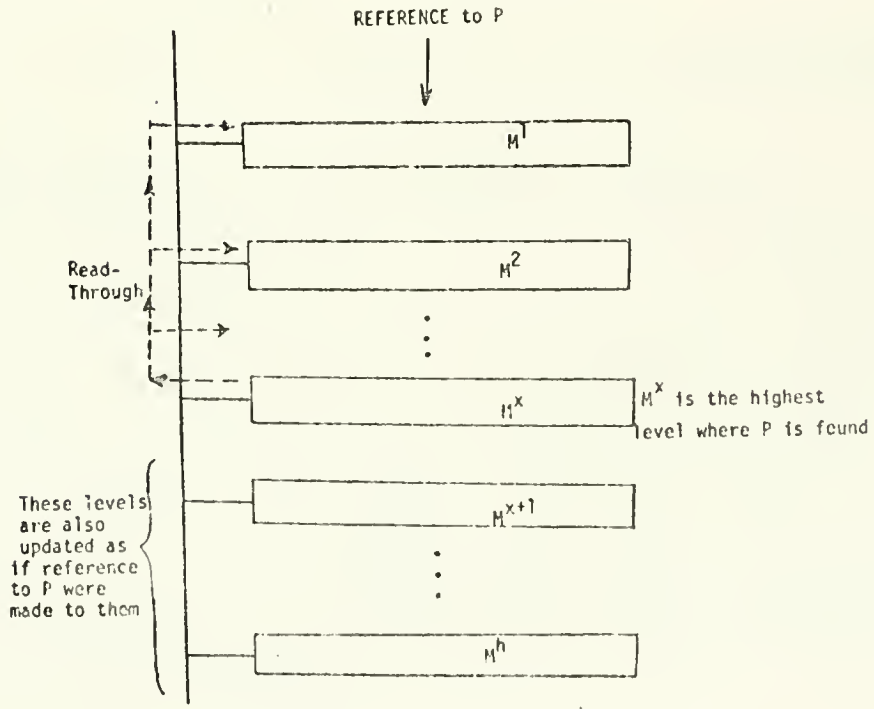

Figure 5 GLOBAL-LRU

2.2 Basic Model of Data Storage Hierarchy

For the purposes of this paper, the basic model illustrated in Figure 6 is sufficient to model the Data Storage Hierarchy. As far as the Read-Through and Overflow-Handling operations are concerned, this basic model is generalizable to a h-level storage hierarchy system.

M^r can be viewed as a reservoir which contains all the information. M^i is the top level. It has m_i pages each of size Q_i. M^j (j=i+1) is the next level. It has m_j pages each of size nQ_i where n is an integer greater than 1.

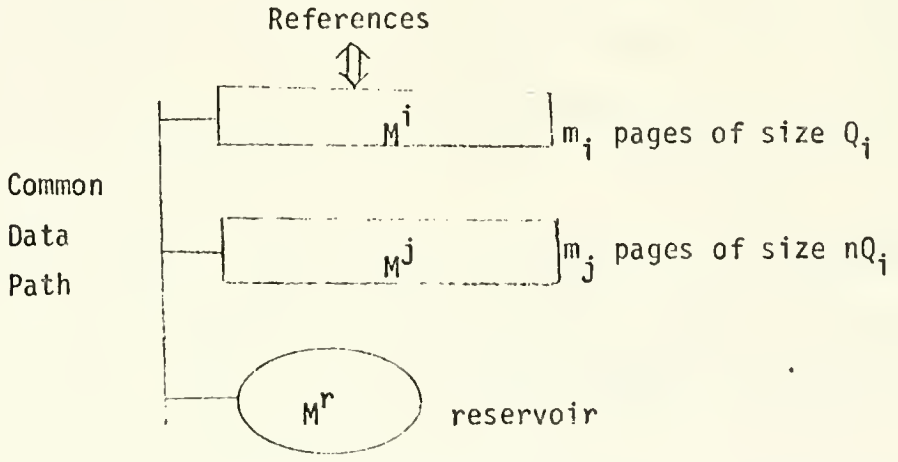

Figure 6 Basic model of a data storage hierarchy

2.3 Formal Definitions of Storage Management Algorithms

Denote a reference string by r = "$r_1, r_2, \ldots r_n$," where $r_t (1 \leq t \leq n)$ is the page being referenced at the t-th reference cycle. Let S_t^i be the stack for M^i at the beginning of the t-th reference cycle, ordered according to LRU. That is, $S_t^i = (S_t^i(1), S_t^i(2), \ldots, S_t^i(K))$, where $S_t^i(1)$ is the most recently referenced page and $S_t^i(K)$ is the least recently referenced page. Note that $K \leq m_i$ (m_i = capacity of M^i in terms of the number of pages). The number of pages in S_t^i is denoted as $|S_t^i|$, hence $|S_t^i| = K$. By convention, $S_1^i = \emptyset$, $|S_1^i| = 0$

S_t^i is an ordered set. Define M_t^i as the contents of S_t^i without any ordering. Similarly, we can define S_t^j and M_t^j for M^j.

Let us denote the pages in M^j by P_1^j, P_2^j, ... Each page, P_y^j, in M^j, consists of an equivalent of n smaller pages, each of size $Q_i = Q_j/n$. Denote this set of pages by $(P_y^j)^i$, i.e., $(P_y^j)^i = \left\{P_{y1}^i, P_{y2}^i, \ldots, P_{yn}^i\right\}$. In general, $(M_t^j)^i$ is the set of pages, each of size Q_i, obtained by " breaking down " the pages in M_t^j. Formally, $(M_t^j)^i = \bigcup_{k=1}^{x} (S_t^j(k))^i$ where $x = \left|S_t^j\right|$. $(P_y^j)^i$ is called the family from the parent page P_y^j. Any pair of pages, P_{ya}^i and P_{yb}^i from $(P_y^j)^i$ are said to be family equivalent, denoted by $P_{ya}^i \overset{f}{=} P_{yb}^i$. Furthermore, a parent page P_y^j and a page P_{yz}^i (for $1 \le z \le n$) from its family are said to be corresponding pages, denoted by $P_{yz}^i \overset{c}{=} P_y^j$.

S_t^i and S_t^j are said to be in corresponding order, denoted by $S_t^i \overset{o}{=} S_t^j$, if $S_t^i(k) \overset{c}{=} S_t^j(k)$ for k = 1, 2, 3, . . . w, where $w = \min\left(\left|S_t^i\right|, \left|S_t^j\right|\right)$. Intuitively, two stacks are in corresponding order if, for each element of the shorter stack, there is a corresponding page in the other stack at the same stack distance (The stack distance for page $S_t^i(k)$ is defined to be k.).

M_t^i and M_t^j are said to be correspondingly equivalent, denoted by $M_t^i \overset{e}{=} M_t^j$ if $\left|M_t^i\right| = \left|M_t^j\right|$ and for any k = 1, 2, . . ., $\left|M_t^i\right|$ there exists x, such that $S_t^i(k) \overset{c}{=} S_t^j(x)$ and $S_t^j(x) \overset{c}{\neq} S_t^i(y)$ for all $y \neq k$. Intuitively, the two memories are correspondingly equivalent when each page in one memory corresponds to exactly one page in the other memory.

The reduced stack, $\bar{S}_t^i$, of S_t^i is defined to be $\bar{S}_t^i(k) = S_t^i(j_k)$ for k = 1, . . ., $\left|\bar{S}_t^i\right|$ where j_k is the minimum j_k where $j_k > j_{k-1}$ ($j_0 = 0$) and $\bar{S}_t^i(k) \overset{f}{\neq} S_t^i(j)$ for $j < j_k$. Intuitively, $\bar{S}_t^i$ is obtained from S_t^i by collecting one page from each family existing in S_t^i, such that the page being collected from each family is the page that has the smallest stack distance within the family.

In the following, we define the storage management algorithms. In each case, assume that the page referenced at time t is P^i_{ya} .

LRU $(S^i_t , P^i_{ya}) = S^i_{t+1}$ is defined as follows:

Case 1 : $P^i_{ya} \in S^i_t$, $P^i_{ya} = S^i_t(k)$:

$$S^i_{t+1}(1) = P^i_{ya} , \quad S^i_{t+1}(x) = \begin{cases} S^i_t(x-1) , & 1 < x \leq k \\ S^i_t(x) , & k < x \leq |S^i_t| \end{cases}$$

Case 2 : $P^i_{ya} \notin S^i_t$:

$S^i_{t+1}(1) = P^i_{ya}$, $S^i_{t+1}(x) = S^i_t(x-1)$, $1 < x \leq \min(m_i, |S^i_t|+1)$

If $|S^i_t| = m_i$ then $P^i_{oa} = S^i_t(m_i)$ is the overflow, else there is no overflow.

LOCAL-LRU-SOP $(S^i_t , S^j_t , P^i_{ya}) = (S^i_{t+1} , S^j_{t+1})$ is defined as follows:

Case 1 : $P^i_{ya} \in S^i_t$:

$S^i_{t+1} =$ LRU (S^i_t , P^i_{ya}) , $S^j_{t+1} = S^j_t$

Case 2 : $P^i_{ya} \notin S^i_t$, $P^j_y \in S^j_t$:

$S^i_{t'} =$ LRU (S^i_t , P^i_{ya}) , $S^j_{t'} =$ LRU (S^j_t , P^j_y),

If there is no overflow from S^i_t

then $S^i_{t+1} = S^i_{t'}$ and $S^j_{t+1} = S^j_{t'}$

If overflow from S^i_t is the page P^i_{oa}

then $(S^i_{t+1} , S^j_{t+1}) =$ SOP $(S^i_{t'} , S^j_{t'} , P^i_{oa})$ defined as :

$S^i_{t+1} = S^i_{t'}$; if $P^j_o \in S^j_{t'}$ then $S^j_{t+1} = S^j_{t'}$,

if $P^j_o \notin S^j_{t'}$ then $S^j_{t+1} =$ LRU $(S^j_{t'} , P^j_o)$

Case 3 : $P^i_{ya} \notin S^i_t$ and $P^j_y \notin S^j_t$:

(handled as in Case 2)

LOCAL-LRU-DOP $(S^i_t , S^j_t , P^i_{ya}) = (S^i_{t+1} , S^j_{t+1})$ is defined as :

Case 1 : $P^i_{ya} \in S^i_t$:

$S^i_{t+1} =$ LRU (S^i_t , P^i_{ya}) , $S^j_{t+1} = S^j_t$

Case 2 : $P^i_{ya} \notin S^i_t$ and $P^j_y \in S^j_t$:

$S^i_{t'} =$ LRU (S^i_t , P^i_{ya}) , $S^j_{t'} =$ LRU (S^j_t , P^j_y)

If no overflow from S^i_t then $S^i_{t+1} = S^i_{t'}$ and $S^j_{t+1} = S^j_{t'}$

If overflow from S_t^i is P_{oa}^i then

$(S_{t+1}^i , S_{t+1}^j) = \underline{DOP}\ (S_{t'}^i , S_{t'}^j , P_{oa}^i)$ which is defined as:

$S_{t+1}^i = S_{t'}^i$ and $S_{t+1}^j = \underline{LRU}\ (S_{t'}^j , P_o^j)$

Case 3 : $P_{ya}^i \notin S_t^i$ and $P_y^j \notin S_t^j$:

(handled as in Case 2 above)

$\underline{GLOBAL\text{-}LRU\text{-}SOP}\ (S_t^i , S_t^j , P_{ya}^i) = (S_{t+1}^i , S_{t+1}^j)$ is defined as follows:

$S_{t'}^i = \underline{LRU}\ (S_t^i , P_{ya}^i)$ and $S_{t'}^j = \underline{LRU}\ (S_t^j , P_y^j)$,

If no overflow from S_t^i then $S_{t+1}^i = S_{t'}^i$ and $S_{t+1}^j = S_{t'}^j$

If overflow from S_t^i is P_{oa}^i then $(S_{t+1}^i , S_{t+1}^j) = \underline{SOP}\ (S_{t'}^i , S_{t'}^j , P_{oa}^i)$

$\underline{GLOBAL\text{-}LRU\text{-}DOP}\ (S_t^i , S_t^j , P_{ya}^i) = (S_{t+1}^i , S_{t+1}^j)$ is defined as:

$S_{t'}^i = \underline{LRU}\ (S_t^i , P_{ya}^i)$ and $S_{t'}^j = \underline{LRU}\ (S_t^j , P_y^j)$

If no overflow from S_t^i then $S_{t+1}^i = S_{t'}^i$ and $S_{t+1}^j = S_{t'}^j$

If overflow from S_t^i is P_{oa}^i then $(S_{t+1}^i , S_{t+1}^j) = \underline{DOP}\ (S_{t'}^i , S_{t'}^j , P_{oa}^i)$

3. Properties of Data Storage Hierarchy

One of the properties of a Read-Through operation is that it leaves a "shadow" of the referenced page (i.e., the corresponding pages) in all storage levels. This provides multiple redundancy for the page. Does this multiple redundancy exist at all times? That is, if a page exists in storage level M^i, will its corresponding pages always be in all storage levels lower than M^i? We refer to this as the Multi-Level Inclusion (MLI) property. As illustrated in Figure 7 for the LOCAL-LRU algorithms and in Figure 8 for the GLOBAL-LRU algorithms, it is not always possible to guarantee that the MLI property holds. For example, after the reference to P^i_{31} in Figure 7(a) the page P^i_{11} exists in M^i but its corresponding page P^j_1 is not found in M^j. In this paper we shall derive the necessary and sufficient conditions for the MLI property to hold at all times.

Another desirable property of the Data Storage Hierarchy is to avoid generating references due to overflows. That is, under what conditions will overflow pages from M^i find their corresponding pages already existing in the storage level M^{i+1}? We refer to this as the Multi-Level Overflow Inclusion (MLOI) property. We shall investigate the conditions that make this property true at all times.

Refering to the basic model of a data storage hierarchy in Figure 6, for high performance it is desirable to minimize the number of references to M^r (the reservoir). If we increased the number of pages in M^i, or in M^j, or in both, we might expect the number of references to M^r to decrease. As illustrated in Figure 9 for the LOCAL-LRU-SOP algorithm, this is not always so, i.e., for the same reference string, the number of references to the reservoir actually increased from 4 to 5

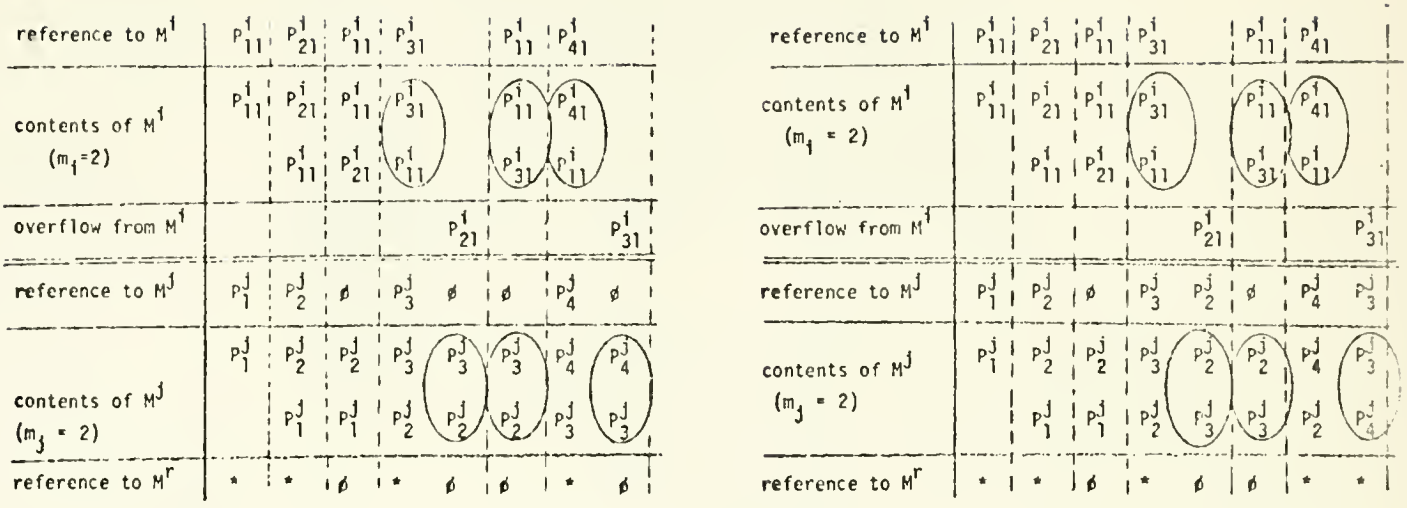

Figure 7 Examples of MLI violations for Local-LRU algorithms

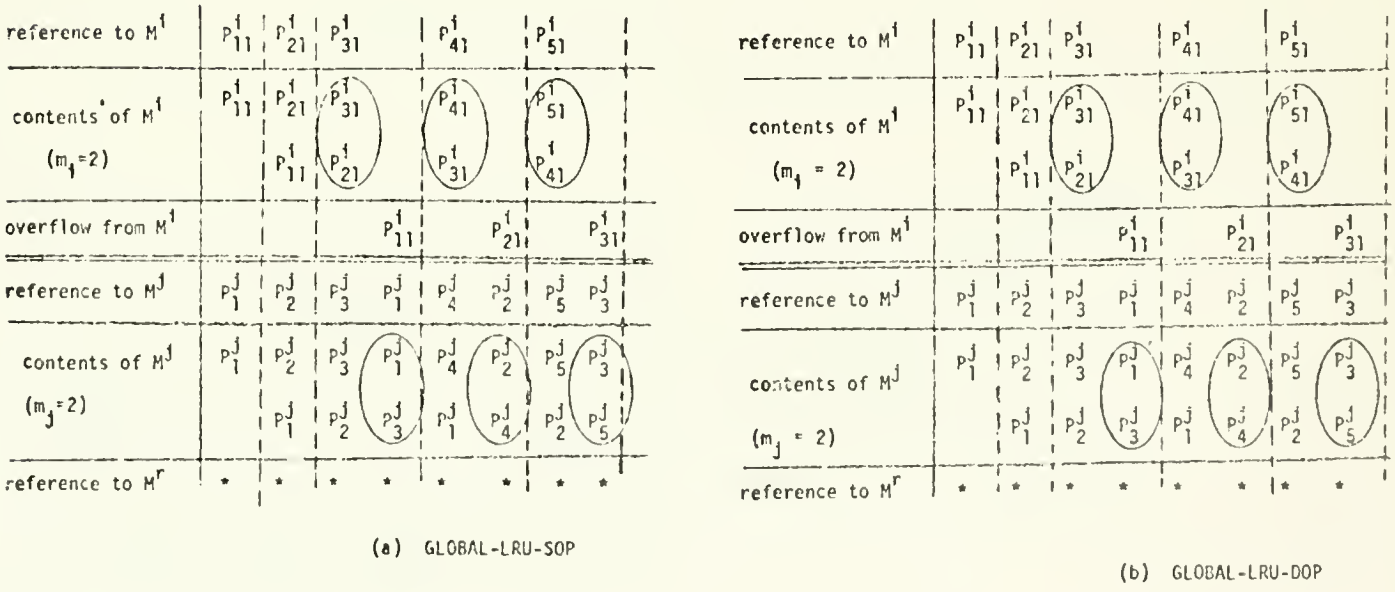

Figure 8 Examples of MLI violations for Global-LRU algorithms

(a) $m_i=2$

reference to M^i	P^i_{11}	P^i_{21}	P^i_{11}	P^i_{31}	P^i_{11}	P^i_{41}
contents of M^i ($m_i = 2$)	P^i_{11}	P^i_{21}	P^i_{11}	P^i_{31}	P^i_{11}	P^i_{41}
		P^i_{11}	P^i_{21}	P^i_{11}	P^i_{31}	P^i_{11}
overflow from M^i				P^i_{21}		P^i_{31} .
reference to M^j	P^j_1	P^j_2	ø	P^j_3 ø	ø	P^j_4 ø
contents of M^j ($m_j = 2$)	P^j_1	P^j_2	P^j_2	P^j_3 P^j_3	P^j_3	P^j_4 P^j_4
		P^j_1	P^j_1	P^j_2 P^j_2	P^j_2	P^j_3 P^j_3
reference to M^r	*	*	ø	* ø	ø	* ø
number of references to M^r = 4						

(b) $m_i=3$

reference to M^i	P^i_{11}	P^i_{21}	P^i_{11}	P^i_{31}	P^i_{11}	P^i_{41}
contents of M^i ($m_i = 3$)	P^i_{11}	P^i_{21}	P^i_{11}	P^i_{31}	P^i_{11}	P^i_{41}
		P^i_{11}	P^i_{21}	P^i_{11}	P^i_{31}	P^i_{11}
				P^i_{21}	P^i_{21}	P^i_{31}
overflow from M^i						P^i_{21}
reference to M^j	P^j_1	P^j_2	ø	P^j_3	ø	P^j_4 P^j_2
contents of M^j ($m_j = 2$)	P^j_1	P^j_2	P^j_2	P^j_3	P^j_3	P^j_4 P^j_2
		P^j_1	P^j_1	P^j_2	P^j_2	P^j_3 P^j_4
reference to M^r	*	*	ø	*	ø	* *
number of references to M^r = 5						

Figure 9 MLPA for LOCAL-LRU-SOP

after M^i is increased by 1 page in size. We refer to this phenomona as a Multi-Level Paging Anomaly (MLPA). One can easily find situations where MLPA occurs for the other three algorithms. Since occurrenceof MLPA reduces performance in spite of the costs of increasing memory sizes, we would like to investigate the conditions to guarantee that MLPA does not exist.

3.1 Summary of Properties

The MLI, MLOI, and MLPA properties of the Data Storage Hierarchy have been derived in the form of eight theorems. These theorems are briefly explained and summarized below and formally proven in the following section.

Multi-Level Inclusion (MLI) : It is shown in Theorem 1 that if the number of pages in M^i is greater than the number of pages in M^j (note M^j pages are larger than those of M^i), then it is not possible to guarantee MLI for all reference strings at all times. It turns out that using LOCAL-LRU-SOP, or LOCAL-LRU-DOP, no matter how many pages are in M^j or M^i, one can always find a reference string that violates the MLI property (Theorem 2). Using the GLOBAL-LRU algorithms, however, conditions to guarantee MLI exist. For the GLOBAL-LRU-SOP algorithm, a necessary and sufficient condition to guarantee that MLI holds at all times for any reference string is that the number of pages in M^j be greater than the number of pages in M^i (Theorem 3). For the GLOBAL-LRU-DOP algorithm, a necessary and sufficient condition to guarantee MLI is that the number of pages in M^j be greater than or equal to twice the number of pages in M^i (Theorem 4).

Multi-Level Overflow Inclusion (MLOI) : It is obvious that if MLI cannot be guaranteed then MLOI cannot be guaranteed. Thus, the LOCAL-LRU algorithms cannot guarantee MLOI. For the GLOBAL-LRU-SOP algorithm, a necessary and sufficient condition to guarantee MLOI is the same condition as that to guarantee MLI (Theorem 5). For the GLOBAL-LRU-DOP algorithm, a necessary and sufficient condition to guarantee MLOI is that the number of pages in M^j is strictly greater than twice the number of pages in M^i (Theorem 6). Thus, for the GLOBAL-LRU-DOP algorithm, guaranteeing that MLOI holds will also guarantee that MLI will hold, but not vice versa.

Multi-Level Paging Anomaly (MLPA) : We have identified and proved sufficiency conditions to avoid MLPA for the GLOBAL-LRU algorithms. For the GLOBAL-LRU-SOP algorithm, this condition is that the number of pages in M^j must be greater than the number of pages in M^i before and after any increase in the sizes of the levels (Theorem 7). For the GLOBAL-LRU-DOP algorithm, this condition is that the number of pages in M^j must be greater than twice the number of pages in M^i before and after any increase in the sizes of the levels (Theorem 8).

In summary, we have shown that for the LOCAL-LRU algorithms, no choice of sizes for the storage levels can guarantee that a lower storage level always contains all the information in the higher storage levels. For the GLOBAL-LRU algorithms, by choosing appropriate sizes for the storage levels, we can (1) ensure that the above inclusion property holds at all times for all reference strings, (2) guarantee that no extra page references to lower storage levels are generated as a result of handling overflows, and (3) guarantee that increasing the sizes of the storage levels does not increase the number of references to lower storage levels. These results are formally stated as the following eight Theorems. Formal proofs of these Theorems are presented in the following section.

THEOREM 1

Under LOCAL-LRU-SOP, or LOCAL-LRU-DOP, or GLOBAL-LRU-SOP, or GLOBAL-LRU-DOP, for any $m_i \geq 2$, $m_j \leq m_i$ implies $\exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$

THEOREM 2

Under LOCAL-LRU-SOP, or LOCAL-LRU-DOP, for any $m_i \geq 2$, and any m_j, $\exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$

THEOREM 3

Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, $\forall$ r,t, $(M_t^j)^i \supseteq M_t^i$ iff $m_j > m_i$

THEOREM 4

Under GLOBAL-LRU-DOP, for any $m_i \geq 2$, $\forall$ r,t, $(M_t^j)^i \supseteq M_t^i$ iff $m_j \geq 2m_i$

THEOREM 5

Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j iff $m_j > m_i$

THEOREM 6

Under GLOBAL-LRU-DOP, for any $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j iff $m_j > 2m_i$

THEOREM 7

Let M^i (with m_i pages), M^j (with m_j pages) and M^r be System A.

Let M'^i (with m_i' pages), M'^j (with m_j' pages) and M^r be System B.

Let $m_i' \geq m_i$ and $m_j' \geq m_j$. Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, no MLPA can exist if $m_j > m_i$ and $m_j' > m_i'$

THEOREM 8

Let System A and System B be defined as in THEOREM 7.

Let $m_i' \geq m_i$ and $m_j' \geq m_j$. Under GLOBAL-LRU-DOP, for any $m_i \geq 2$, no MLPA can exist if $m_j > 2m_i$ and $m_j' > 2m_i'$

3.2 Derivation of Properties

THEOREM 1

Under LOCAL-LRU-SOP, or LOCAL-LRU-DOP, or GLOBAL-LRU-SOP, or GLOBAL-LRU-DOP, for any $m_i \geq 2$, $m_j \leq m_i$ implies $\exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$

PROOF

Case 1 : $m_j < m_i$

Consider the reference string r=" P_{1a}^i , P_{2a}^i , ... , $P_{(m_j+1)a}^i$ ".

Using any one of the algorithms, the following stacks are obtained at $t=m_j+2$:

$S_t^i = (P_{(m_j+1)a}^i , P_{m_j a}^i , ... , P_{2a}^i , P_{1a}^i)$

$S_t^j = (P_{(m_j+1)}^j , P_{m_j}^j , ... , P_3^j , P_2^j)$

Thus, $P_{1a}^i \in M_t^i$ but $P_{1a}^i \notin (M_t^j)^i$, i.e., $(M_t^j)^i \not\supseteq M_t^i$.

Case 2 : $m_j = m_i = w$

Consider the reference string r = " P_{1a}^i , P_{2a}^i , ... , $P_{(w+1)a}^i$ "

Using any one of the above algorithms, the following stacks are obtained at t=w+2 :

$S_t^i = (P_{(w+1)a}^i , P_{wa}^i , ... , P_{3a}^i , P_{2a}^i)$

$S_t^j = (P_1^j , P_{(w+1)}^j , P_w^j , ... , P_4^j , P_3^j)$

Thus, $P_{2a}^i \in M_t^i$ but $P_{2a}^i \notin (M_t^j)^i$, i.e., $(M_t^j)^i \not\supseteq M_t^i$.

Q.E.D.

THEOREM 2

Under LOCAL-LRU-SOP, or LOCAL-LRU-DOP, for any $m_i \geq 2$, and any m_j ,

$$\exists \; r,t, \; (M_t^j)^i \not\supseteq M_t^i \; .$$

PROOF (For LOCAL-LRU-SOP)

For $m_j \leq m_i$ the result follows directly from THEOREM 1.

For $m_j > m_i$, using the reference string

$r = " \; P^i_{za} \; , \; P^i_{1a} \; , \; P^i_{za} \; , \; P^i_{2a} \; , \; . \; . \; . \; , \; P^i_{za} \; , \; P^i_{m_j a} \; "$,

the following stacks will be produced at $t=2m_j+1$:

$S^i_t = (\; P^i_{m_j a} \; , \; P^i_{za} \; , \; P^i_{(m_j-1)a} \; , \; . \; . \; . \; , \; P^i_{(m_j-m_i+2)a} \;)$

$S^j_t = (\; P^j_{m_j} \; , \; P^j_{m_j-1} \; , \; . \; . \; . \; , \; P^j_2 \; , \; P^j_1 \;)$

Thus $P^i_{za} \in M^i_t$ but $P^i_{za} \notin (M^j_t)^i$, i.e., $(M^j_t)^i \not\supseteq M^i_t$. Q.E.D.

PROOF (For LOCAL-LRU-DOP)

For $m_j \leq m_i$ the result follows directly from THEOREM 1.

For $m_j > m_i$, using the following reference string

$r = " \; P^i_{za} \; , \; P^i_{1a} \; , \; P^i_{za} \; , \; P^i_{2a} \; , \; . \; . \; . \; , \; P^i_{za} \; , \; P^i_{m_j a} \; "$,

The following stacks will be produced at $t=2m_j+1$:

$S^i_t = (\; P^i_{m_j a} \; , \; P^i_{za} \; , \; . \; . \; . \; , \; P^i_{(m_j-m_i+2)a} \;)$, $S^j_t = (\; a_1 \; , \; a_2 \; , \; . \; . \; . \; , \; a_{m_j} \;)$

Where for $1 \leq i \leq m_j$, $a_i \in \left\{ P^j_{m_j} \; , \; P^j_{m_j-1} \; , \; \ldots \; , \; P^j_3 \; , \; P^j_2 \; , \; P^j_1 \right\}$

since P^j_z is the only overflow from M^j.

Thus, $P^i_{za} \in M^i_t$ but $P^i_{za} \notin (M^j_t)^i$, i.e., $(M^j_t)^i \not\supseteq M^i_t$.

Q.E.D.

THEOREM 3

Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, $\forall$ r,t, $(M_t^j)^i \supseteq M_t^i$ iff $m_j > m_i$

PROOF

This proof has two parts. Part (a) to prove $\forall$ r,t, $(M_t^j)^i \supseteq M_t^i \Rightarrow m_j > m_i$
or equivalently, $m_j \leq m_i \Rightarrow \exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$
Part (b) to prove $m_j > m_i \Rightarrow \forall$ r,t, $(M_t^j)^i \supseteq M_t^i$

PROOF of Part (a) : $m_j \leq m_i \Rightarrow \exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$

This follows directly from THEOREM 1.

Q.E.D.

To prove Part (b), we need the following results.

LEMMA 3.1

$\forall$ r, t such that $|M_t^j| \leq m_i$, if $m_j = m_i + 1$, then
(a) $(M_t^j)^i \supseteq M_t^i$, and (b) $\bar{S}_t^i \subseteq S_t^j$

PROOF of LEMMA 3.1

For t=2 (i.e., after the first reference), (a) and (b) are true.

Suppose (a) and (b) are true for t, such that $|M_t^j| \leq m_i$

Consider the next reference :

Case 1 : It is a reference to M^i :

There is no overflow from M^i or M^j, so (a) is still true.

Since Global-LRU is used, (b) is still true.

Case 2 : It is a reference to M^j :

There is no overflow from M^j. If no overflow from M^i, the same arguement as Case 1 applies. If there is overflow from M^i, the overflow page finds its corresponding page in M^j. Since SOP is used, this overflow can be treated as a "no-op". Thus (a) and (b) are preserved.

Case 3 : It is a reference to M^r :

There is no overflow from M^j since $|M_{t+1}^j| \leq m_i$. Thus the same reasoning as in Case 2 applies.

Q.E.D.

LEMMA 3.2

$\forall$ r, t, such that $|M_t^j| = m_j$, if $m_j = m_i + 1$ then

(a) $(M_t^j)^i \supset M_t^i$, (b) $\bar{S}_t^i \subseteq S_t^j$, and (c) $(S_t^j(m_j))^i \cap S_t^i = \emptyset$

Let us denote the conditions (a) (b) and (c) jointly as Z(t).

PROOF of LEMMA 3.2

Suppose the first time $S_t^j(m_j)$ is filled is by the t*-th reference. That is, $S_t^j(m_j) = \emptyset$ for all $t \leq t^*$ and $S_t^j(m_j) \neq \emptyset$ for all $t > t^*$. From LEMMA 3.1 we know that (a) and (b) are true for all $t \leq t^*$. Let $t_1 = t^* + 1$, $t_2 = t^* + 2$, ..., etc. We shall show, by induction on t, starting at t_1 , that Z(t) is true. First we show that $Z(t_1)$ is true as follows:

Case 1 : $M_{t^*}^j \subseteq M_{t^*}^i$

$\bar{S}_{t^*}^i \subseteq S_{t^*}^j$ and $M_{t^*}^j \subseteq M_{t^*}^i \Rightarrow S_{t^*}^j(m_j - 1) \subseteq S_{t^*}^i(m_i)$

As a result of the reference at t* (to M^r), $S_{t^*+1}^j(m_j) = S_{t^*}^j(m_j - 1)$ and $S_{t^*}^i(m_i)$ overflows from M^i. This overflow page finds its corresponding page in M^j because there is no overflow from M^j and (a). Since SOP is used, the overflow from M^i can be treated as a "no-op". Furthermore, since Global-LRU is used, (b) is true after the t*-th reference. (b) and $|S_{t^*+1}^j| > |S_{t^*+1}^i| \Rightarrow$(a) and (c). Thus $Z(t_1)$ is true.

Case 2 : $(M_{t^*}^j)^i \supset M_{t^*}^i$ and $M_{t^*}^j \not\subseteq M_{t^*}^i$

$(M_{t^*}^j)^i \supset M_{t^*}^i$ and $M_{t^*}^j \not\subseteq M_{t^*}^i \Rightarrow \exists\ S_{t^*}^j(k)$ such that $(S_{t^*}^j(k))^i \cap M_{t^*}^i = \emptyset$

$\bar{S}_{t^*}^i \subseteq S_{t^*}^j$ and $(S_{t^*}^j(k))^i \cap M_{t^*}^i = \emptyset \Rightarrow k > |\bar{S}_{t^*}^i|$ and $(S_{t^*}^j(x))^i \cap M_{t^*}^i = \emptyset$ for all x, where $m_{j-1} \geq x \geq k$. Thus $(S_{t^*}^j(m_{j-1}))^i \cap S_{t^*}^i = \emptyset$ (i.e., the last page of $S_{t^*}^j$ is not in $S_{t^*}^i$)

$S_{t^*}^i(m_i)$ overflows from M^i. There is no overflow from M^j. Thus the overflow page from M^i finds its corresponding page in M^j. For the same reasons as in Case 1, (b) is still preserved. (b) and $|S_{t^*+1}^j| > |S_{t^*+1}^i| \Rightarrow$(a) and (c) are true. Thus, $Z(t_1)$ is true.

Assume that $Z(t_k)$ is true; to show that $Z(t_{k+1})$ is true, we consider the next reference, at time t_{k+1} :

Imagine that the last page of $S^j_{t_k}$ does not exist, i.e., $S^j_{t_k}(m_j) = \emptyset$

If the reference at t_{k+1} is to a page in $M^i_{t_k}$ or $M^j_{t_k}$, then (a) and (b) still hold because Global-LRU is used and because overflow from M^i finds its corresponding page in M^j (See the proof of LEMMA 3.1).

If the reference at t_{k+1} is to a page not in $M^j_{t_k}$, then we can apply the argument as that used in considering the reference at time t_1 above to show that $Z(t_{k+1})$ is still true.

Q.E.D.

LEMMA 3.3

$\forall$ r,t, if $m_j = m_i + 1$ then (a) $(M^j_t)^i \supseteq M^i_t$ and (b) $(S^j_t(m_j))^i \cap S^i_t = \emptyset$

PROOF of LEMMA 3.3

For t such that $|M^j_t| \leq m_i$ (a) follows directly from LEMMA 3.1 and (b) is true because $S^j_t(m_j) = \emptyset$

For t such that $|M^j_t| = m_j$ (a) and (b) follows directly from LEMMA 3.2

Q.E.D.

LEMMA 3.4

$\forall$ r,t, if $m_j > m_i$ then (a) $(M^j_t)^i \supseteq M^i_t$ and (b) $(S^j_t(m_j))^i \cap S^i_t = \emptyset$

PROOF of LEMMA 3.4

Let $m_j = m_i + k$. We shall prove this lemma by induction on k.

For k=1 (a) and (b) are true from LEMMA 3.3.

Suppose that (a) and (b) are true for k.

Consider $m_j = m_i + (k+1)$. That is consider the effects of increasing M^j by 1 page in size :

Since M^i is unchanged, M^j (with m_i+k+1 pages) sees the same reference string as M^j (with m_i+k pages). Applying the stack inclusion property (Mattson.et al., 70), we have $M^j(\text{with } m_i+k+1 \text{ pages}) \supseteq M^j(\text{with } m_i+k \text{ pages})$. Thus (a) is still true. Suppose $(S_t^j(m_i+k+1))^i \bigcap S_t^i \neq \emptyset$ then there is a page in M^i that corresponds to this page. But $S_t^j(m_i+k+1)$ is not in M^j (with m_i+k pages). This contradicts the property that $(M_t^j)^i \supseteq M_t^i$. This shows that (b) is still true.

Q.E.D.

PROOF of Part(b) : $m_j > m_i \Rightarrow \forall\ r,t,\ (M_t^j)^i \supseteq M_t^i$:

This follows directly from LEMMA 3.4.

Q.E.D.

THEOREM 4

Under GLOBAL-LRU-DOP, for any $m_i \geq 2$, $\forall r,t$, $(M_t^j)^i \supseteq M_t^i$ iff $m_j \geq 2m_i$

PROOF

This proof has two parts:

Part (a) : $m_j < 2m_i \Rightarrow \exists r,t,\ (M_t^j)^i \not\supseteq M_t^i$

Part (b) : $m_j \geq 2m_i \Rightarrow \forall\, r,t,\ (M_t^j)^i \supseteq M_t^i$

PROOF of Part (a) : $m_j < 2m_i \Rightarrow \exists\, r,t,\ (M_t^j)^i \not\supseteq M_t^i$

For $m_j \leq m_i$ the result follows from THEOREM 1.

Consider the case for $2m_i > m_j > m_i$:

The reference string r = " P^i_{1a} , P^i_{2a} , P^i_{3a} , . . . , $P^i_{(2m_i)a}$ "

will produce the following stacks:

$S_t^i = (P^i_{(2m_i)a} , P^i_{(2m_i-1)a} , \ldots , P^i_{(m_i+1)a})$,

$S_t^j = (a_1 , a_2 , a_3 , \ldots , a_{m_j})$ where a_i's are picked from L_1 and L_2 alternatively, starting from L_1. $L_1 = (P^j_{m_i} , P^j_{(m_i-1)}, \ldots , P^j_1)$ and $L_2 = (P^j_{2m_i} , P^j_{(2m_i-1)}, \ldots , P^j_{m_i+1})$.

If m_j is even, then $(a_1 , a_3 , \ldots a_{m_j-1})$ corresponds to the first $m_j/2$ elements of L_1 and $(a_2 , a_4 , \ldots a_{m_j})$ corresponds to the first $m_j/2$ elements in L_2. We see that $P^i_{(m_i+1)a}$ is in S_t^i but its corresponding page is not in S_t^j ($P^j_{(m_i+1)}$ is not in S_t^j since $m_j/2 < m_i$).

If m_j is odd, then $(a_1 , a_3 , \ldots a_{m_j})$ corresponds to the first $(m_j+1)/2$ elements in L_1 and $(a_2 , a_4, \ldots, a_{m_j-1})$ corresponds to the first $(m_j-1)/2$ elements in L_2. We see that the page $P^i_{(m_i+1)a}$ is in S_t^i but its corresponding page is not in S_t^j because $\max((m_j-1)/2) = m_i-1$, thus, $a_{(m_j-1)}$ is at most the (m_i-1)-th element of L_2, $P^j_{2m_i-(m_i-1)+1} = P^j_{m_i+2}$.

In both cases, $(M_t^j)^i \not\supseteq M_t^i$

Q.E.D.

To prove Part (b), we need the following preliminary results.

LEMMA 4.1

Under GLOBAL-LRU-DOP, for $m_i \geq 2$, $m_j \geq 2m_i$, a page found at stack distance k in M_t^i implies its corresponding page can be found within stack distance 2k in M_t^j.

PROOF of LEMMA 4.1

We prove by induction on t.

At t=1, the statement is trivially true. At t=2 (i.e., after the first reference) $S_t^i(1)$ and its corresponding page are both at the beginning of the stack, hence the induction statement is still true.

Suppose the induction statement is true at time t, i.e.,
$P_{za}^i = S_t^i(k) \Rightarrow P_z^j$ can be found within stack distance 2k within S_t^j.

Suppose the next reference is to P_{wa}^i. There are three cases :

Case 1 : $P_{wa}^i \in M_t^i$ $(P_{wa}^i = S_t^i(x))$

From the induction statement, P_w^j is found within stack distance 2k in S_t^j as illustrated in Figure 10.

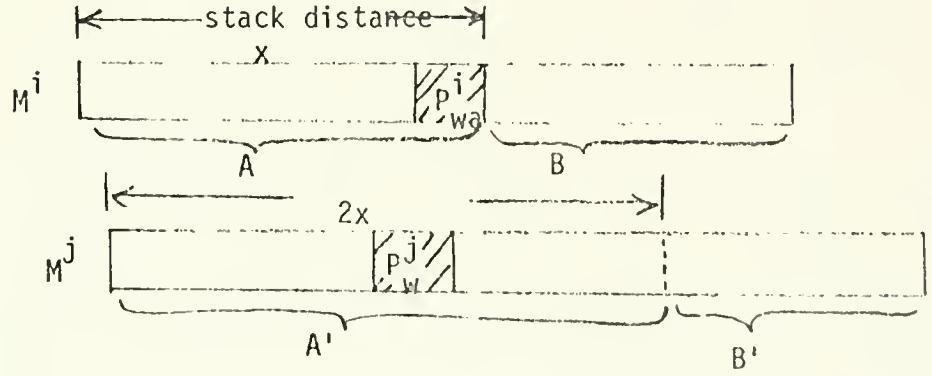

Figure 10.

Consider the page movements in the two stacks as a result of handling the reference to P_{wa}^i :

(1) P_{wa}^i and P_w^j are both moved to the top of their stack, the induction

(2) Each page in A increases its stack distance by 1, but its corresponding page is in A', each page of which can at most increase its stack distance by 1. Thus the induction statement holds for all pages in A.

(3) None of the pages in B are moved. None of the pages in B' are moved. (See previous diagram) If a page in B has its corresponding page in B', the induction statement is not violated. Suppose a page in B, $P^i_{ba} = S^i_t(k)$ $(k>x)$, has its corresponding page, $P^j_b = S^j_t(w)$ in A'. Then P^j_b can at most increase its stack distance by 1. But $w \leq 2x$ because $P^j_b \in A'$. Since $2k > 2x$, the induction statement is not violated.

Case 2 : $P^i_{wa} \notin M^i_t$, $P^j_w \in M^j_t$

Each page in M^i increases its stack distance by 1. Each corresponding page in M^j can at most increase its stack distance by 2, one due to the reference and one due to an overflow from M^i. Hence if $P^i_{za} = S^i_t(k)$, $k < m_i$, then $P^i_{za} = S^i_{t+1}(k+1)$, and P^j_z can be found within stack distance $2(k+1)$ in M^j at time $t+1$.

Case 3 : $P^i_{wa} \notin M^i_t$, $P^j_w \notin M^j_t$

As a result of the read-through from M^r , each page in M^i is increased by a stack distance of 1. That is, for $k < m_i$, $P^i_{za} = S^i_t(k) \Rightarrow P^i_{za} = S^i_{t+1}(k+1)$.

Each page in M^j can at most increase its stack distance by 2, one due to loading the referenced page and one due to an overflow from M^i. Hence, the page P^j_z is found within stack distance of $2k+2$ in M^j. Since $\max(2k+2) = 2m_i \leq m_j$, P^j_z is still in M^j

Q.E.D.

COROLLARY to LEMMA 4.1

$$m_j > 2m_i \Rightarrow \forall\ r,t,\quad (S_t^j(m_j))^i \cap S_t^i = \emptyset$$

PROOF of COROLLARY

For any P_{za}^i in S_t^i, its corresponding page can be found within stack distance $2m_i$ in S_t^j, and since pages in S_t^j are unique, the information in the last page of S_t^j is not found in S_t^i, i.e., $(S_t^j(m_j))^i \cap S_t^i = \emptyset$.

PROOF of Part (b) : $m_j \geq 2m_i \Rightarrow \forall\, r,t,\ (M_t^j)^i \supseteq M_t^i$

This follows directly from LEMMA 4.1.

Q.E.D.

THEOREM 5

Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j iff $m_j > m_i$

COROLLARY

Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j iff $\forall$ r,t, $(M_t^j)^i \supseteq M_t^i$

PROOF

This Proof has two parts as shown below.

PROOF of Part (a) : $m_j > m_i \Rightarrow \forall$ r,t, an overflow from M^i finds its corresponding page in M^j

From LEMMA 3.4 $m_j > m_i \Rightarrow \forall$ r,t, $(M_t^j)^i \supseteq M_t^i$ and $(S_t^j(m_j))^i \cap S_t^i = \phi$

Suppose the overflow from M^i, P_{oa}^i is caused by a reference to M^j.

Then just before P_{oa}^i is overflowed, P_o^j exists in M^j

After the overflow, P_{oa}^i finds its corresponding page still existing in M^j

Suppose the overflow, P_{oa}^i, is caused by a reference to M^r.

Then just before the overflow from M^i, P_o^j exists in M^j and $(S_t^j(m_j))^i \cap S_t^i = \phi$

i.e., the information in the last page of M^j is not in M^i. This means that the last page of M^j is not P_o^j, thus, the overflow page P_{oa}^i finds its corresponding page still in M^j after an overflow from M^j occurs.

PROOF of Part (b) : $m_j \leq m_i \Rightarrow \exists$ r,t, such that an overflow from M^i does not find its corresponding page in M^j

From THEOREM 1, $m_j \leq m_i \Rightarrow \exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$, then there exists $P_{za}^i \in M_t^i$ and $P_z^j \notin M_t^j$. We can find a reference string such that at the time of the overflow of P_{za}^i from M^i, P_z^j is still not in M^j.

A string of references to M^r will produce this condition.

Then at the time of overflow of P_{za}^i, it will not find its corresponding page it M^j.

Q.E.D.

THEOREM 6

Under GLOBAL-LRU-DOP, for $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j iff $m_j > 2m_i$

COROLLARY

Under GLOBAL-LRU-DOP, for $m_i \geq 2$, $\forall$ r,t, an overflow from M^i finds its corresponding page in M^j implies that $\forall$ r,t, $(M_t^j)^i \supseteq M_t^i$

PROOF

This Proof has two parts as shown below.

PROOF of Part (a) : $m_j > 2m_i \Rightarrow \forall$r,t, an overflow from M^i finds its corresponding page it M^j

THEOREM 4 ensures that $m_j > 2m_i \Rightarrow \forall r,t$, $(M_t^j)^i \supseteq M_t^i$ and LEMMA 4.1 ensures that $(S_t^j(m_j))^i \cap S_t^i = \emptyset$, we then use the same argument as in Part (a) of THEOREM 5.

PROOF of Part (b) : $m_j \leq 2m_i \Rightarrow \exists$ r,t, such that an overflow from M^i does not find its corresponding page in M^j

Case 1 : $m_j < 2m_i$

$m_j < 2m_i \Rightarrow \exists$ r,t, $(M_t^j)^i \not\supseteq M_t^i$ (from the proof of part(a) of THEOREM 4).

We then use the same argument as in Part (b) of THEOREM 5.

Case 2 : $m_j = 2m_i$

The reference string r = " P^i_{1a} , P^i_{2a} , . . . , $P^i_{(2m_i)a}$, $P^i_{(2m_i+1)a}$ "

will produce the following stacks (at $t=2m_i+1$):

$S_t^i = (P^i_{(2m_i)a}$, $P^i_{(2m_i-1)a}$, . . . , $P^i_{(m_i+1)a})$

$S_t^j = (P^j_{m_i}$, $P^j_{2m_i}$, $P^j_{m_i-1}$, $P^j_{2m_i-1}$, . . . , P^j_1 , $P^j_{m_i+1})$

In handling the next reference, to page $P^i_{(2m_i+1)a}$, the pages $P^i_{(m_i+1)a}$ and $P^j_{m_i+1}$ overflow at the same time, hence the overflow page $P^i_{(m_i+1)a}$ from M^i does not find its corresponding page in M^j

Q.E.D.

THEOREM 7

Let M^i (with m_i pages), M^j (with m_j pages) and M^r be System A.
Let M'^i (with m_i' pages), M'^j (with m_j' pages) and M^r be System B.
Let $m_i' \geq m_i$ and $m_j' \geq m_j$. Under GLOBAL-LRU-SOP, for any $m_i \geq 2$, no MLPA can exist if $m_j > m_i$ and $m_j' > m_i'$

PROOF

We shall show that $\forall r,t$, $(M_t^i \cup (M_t^j)^i) \subseteq (M'^i_t \cup (M'^j_t)^i)$
This will ensure that no MLPA can exist.
Since $m_i' \geq m_i$ and LRU is used in M^i and M'^i, we can apply the LRU stack inclusion property to obtain $M_t^i \subseteq M'^i_t$.
From THEOREM 5, we know that overflows from M^i or from M'^i always find their corresponding pages in M^j and M'^j respectively. Since SOP is used, these overflows can be treated as "no-ops".
Thus, M^j and M'^j see the same reference string and we can apply the LRU stack inclusion property to obtain $M_t^j \subseteq M'^j_t$ (since $m_j' \geq m_j$ and LRU is used).
$M_t^i \subseteq M'^i_t$ and $M_t^j \subseteq M'^j_t \Rightarrow (M_t^i \cup (M_t^j)^i) \subseteq (M'^i_t \cup (M'^j_t)^i)$

Q.E.D.

THEOREM 8

Let System A and System B be defined as in THEOREM 7.

Let $m_i' \geq m_i$ and $m_j' \leq m_j$. Under GLOBAL-LRU-DOP, for any $m_i \geq 2$, no MLPA can exist if $m_j > 2m_i$ and $m_j' > 2m_i'$.

PROOF

We need the following preliminary results for this proof.

LEMMA 8.1

Let S_t^j be partitioned into two disjoint stacks, W_t and V_t defined as follows: $W_t(k) = S_t^j(j_k)$ for $k=1,\ldots,|W_t|$ where $j_0=0$, and j_k is the minimum $j_k > j_{k-1}$ such that $\exists\ P_{za}^i \in S_t^i$ and $P_{za}^i \subseteq S_t^j(j_k)$. $V_t(k) = S_t^j(j_k)$ for $k=1,\ldots,|V_t|$ where $j_0=0$, and j_k is the minimum $j_k > j_{k-1}$ such that $\forall\ P_{za}^i \in S_t^i$, $P_{za}^i \not\subseteq S_t^j(j_k)$. (Intuitively, W_t is the stack obtained from S_t^j by collecting those pages that have their corresponding pages in M_t^i such that the order of these pages in S_t^j is preserved. V_t is what is left of S_t^j after W_t is formed.) Then, $\forall$ r,t, (a) $W_t \subseteq \overline{S}_t^i$ and (b) $V_t \subseteq O_t$ where O_t is the set of pages corresponding to all the pages that ever overflowed from M^i , up to time t.

PROOF of LEMMA 8.1

From THEOREM 4, $m_j > 2m_i \Rightarrow \forall r,t,\ (M_t^j)^i \supseteq M_t^i$. Thus, for each page in M_t^i , its corresponding page is in M_t^j . This set of pages in M_t^j is exactly W_t , and $W_t \subseteq \overline{S}_t^i$ by definition. Since the conditions for V_t and W_t are mutually exclusive and collectively exhaustive, the other pages in M_t^j that are not in W_t are by definition in V_t . Since a page in V_t does not have a corresponding page in M_t^i , its corresponding page must have once been in M^i because of Read-Through, and later overflowed from M^i . Thus a page in V_t is a page in O_t .

Q.E.D.

LEMMA 8.2

Any overflow page from M_t^j is a page in V_t

PROOF of LEMMA 8.2

From THEOREM 4, $m_j > 2m_i \Rightarrow \forall$ r,t, $(M_t^j)^i \supseteq M_t^i$

From THEOREM 6, $m_j > 2m_i \Rightarrow \forall$ r,t, an overflow from M^i always finds its corresponding page in M^j

An overflow from M_t^j is caused by a reference to M^r. An overflow from M_t^j also implies that there is an overflow from M_t^i.

Suppose the overflow page from M_t^j is P_o^j. Also suppose $P_o^j \in W_t$, i.e., $P_o^j \notin V_t$. We shall show that this leads to a contradiction.

The overflow page from M_t^i is either P_{oa}^i or P_{ya}^i $(y \neq o)$.

If $P_{oa}^i \subseteq P_o^j$ is overflowed from M_t^i, THEOREM 6 is violated since P_{oa}^i and P_o^j overflow at the same time so P_{oa}^i will not find its corresponding page in M^j.

If $P_{ya}^i \subsetneq P_o^j$ is overflowed from M_t^i, THEOREM 4 is violated since after the overflow handling, there exists a page $P_{ob}^i \subseteq P_o^j$ in M^i (since $P_o^j \in W_t$) but P_o^j is no longer in M^j.

Q.E.D.

LEMMA 8.3

If there is no overflow from either M^j or M'^j then $\forall$ r,t, V_t and V'_t have the same reverse ordering.

Two stacks S^i and S^j are in the same reverse ordering, $S^i \underline{ro} S^j$, if $rS^i(k) = rS^j(k)$ for $1 \leq k \leq \min(|S^i|, |S^j|)$, where rS denotes the stack obtained from S by reversing its ordering. By convention, $S^i \underline{ro} S^j$ if $S^i = \emptyset$ or $S^j = \emptyset$

PROOF of LEMMA 8.3

To facilitate the proof, we introduce the following definitions.

(1) The ordered parent stack, $(S^i)^j$, of the stack S^i is the stack of parent pages corresponding to, and in the same ordering as, the pages in the reduced stack, $\bar{S}^i$, of S^i. Formally, $(S^i)^j \subseteq \bar{S}^i$ and $(S^i)^j \underline{o} \bar{S}^i$

(2) Define a new binary operator, concatenation ($||$), between two stacks, S^1 and S^2, to produce a new stack, S, as follows:

$S = S^1 \,||\, S^2$, where $S(k) = \begin{cases} S^1(k) \text{ for } k=1,2,\ldots,|S^1| \\ S^2(k) \text{ for } k=|S^1|+1, \ldots,\{|S^1| + |S^2|\} \end{cases}$

(3) Define a new binary operator, ordered difference ($\underline{o}$), between a stack S^1 and a set T, to produce a new stack, S, as follows:

$S = S^1 \,\underline{o}\, T$, where $S(k)=S^1(j_k)$ for $k=1,2,\ldots,(|S^1| - |S^1 \cap T|)$, such that $j_0=0$, j_k is the minimum $j_k > j_{k-1}$ such that $S^1(j_k) \cap T = \emptyset$. Intuitively, S is obtained from S^1 by taking away those elements of S^1 which are also in T.

Figure 11 illustrates the LRU ordering of all Level i pages ever referenced up to time t. Since there is no overflow from either M^j or M'^j, the length of this LRU stack is less than or equal to $\min(m_j, m_j')$

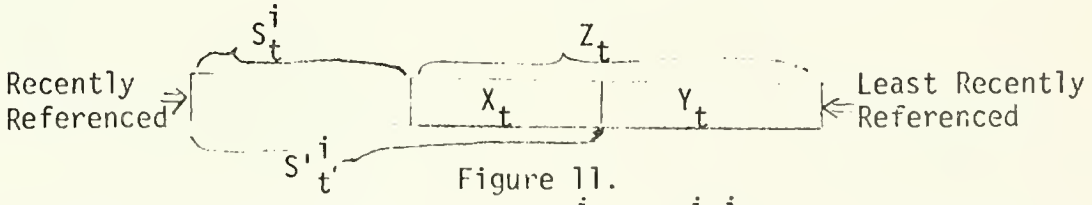

Figure 11.

By the definition of V_t', $V_t' = (Y_t)^j \,\underline{o}\, (S'^i_t)^j$

But $(S'^i_t)^j = (S^i_t)^j \,||\, ((X_t)^j \,\underline{o}\, (S^i_t)^j)$,

hence $V_t' = (Y_t)^j \,\underline{o}\, ((S^i_t)^j \,||\, ((X_t)^j \,\underline{o}\, (S^i_t)^j)) = (Y_t)^j \,\underline{o}\, ((S^i_t)^j \cup (X_t)^j)$

Similarly, by the definition of V_t, $V_t = (Z_t)^j \,\underline{o}\, (S^i_t)^j$

But $(Z_t)^j = (X_t)^j \,||\, ((Y_t)^j \,\underline{o}\, (X_t)^j)$,

hence $V_t = ((X_t)^j \,\underline{o}\, (S^i_t)^j) \,||\, (((Y_t)^j \,\underline{o}\, (X_t)^j) \,\underline{o}\, (S^i_t)^j)$

$= ((X_t)^j \,\underline{o}\, (S^i_t)^j) \,||\, ((Y_t)^j \,\underline{o}\, ((S^i_t)^j \cup (X_t)^j)) = ((X_t)^j \,\underline{o}\, (S^i_t)^j) \,||\, V_t'$

Thus, the two stacks are in the same reverse ordering. Q.E.D.

LEMMA 8.4

$\forall r,t$, (a) $M'^{j}_{t} \supseteq M^{j}_{t}$, (b) V_t and V'_t are either in the same reverse ordering or the last element of V'_t is not an element of V_t

PROOF of LEMMA 8.4

(a) and (b) are true for any time before there is any overflow from either M^j or M'^j. (a) is true because any page ever referenced is in Level j, so a page found in M^j is also found in M'^j. (b) is true because of the result from LEMMA 8.3.

Assume that (a) and (b) is true for t. Consider the next reference at t+1.

Suppose this reference does not produce any overflow from either M^j or M'^j, then (a) still holds because $M'^{j}_{t} \supseteq M^{j}_{t}$ and $M'^{i}_{t} \supseteq M^{i}_{t}$ (See THEOREM 7). (b) still holds because overflows from M^j and M'^j are taken from the end of stacks V_t and V'_t respectively, and since there is no overflow from Level j, (b)'s validity is not disturbed.

Suppose this reference does produce overflow(s) from Level j.

Case 1 : overflow from M'^j , no overflow from M^j :

This cannot happen since overflow from M'^j implies reference to M^r which in turn implies overflow from M^j also.

Case 2 : overflow from M^j , no overflow from M'^j :

Suppose the last element in V'_t is not an element of V_t. Then starting from the end of V'_t , if we eliminate those elements not in V_t , the two stacks will be in the same reverse ordering. This follows from LEMMA 8.3 and is illustrated in Figure 12.

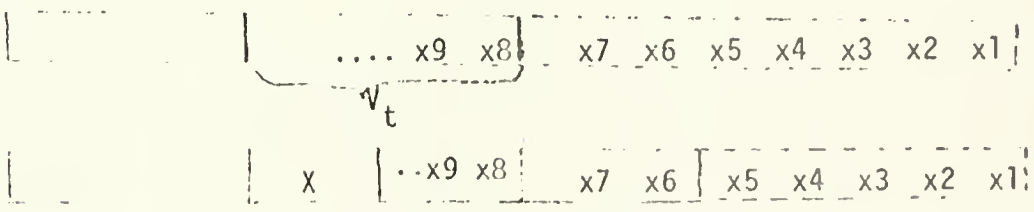

Figure 12.

Thus we see that overflow from M^j, i.e., overflowing the last page of V_t, will not violate (a) since this page is still in V'_t.
(b) is still preserved since the last page in V'_t is still not in V_t.

- Suppose V'_t and V_t are in the same reverse ordering. Then overflowing the last page of V_t does not violate (a) and results in the last page of V'_t not in V_t.

Case 3 : overflow from M^j and overflow from M'^j :

- Suppose the last element in V'_t is not in V_t. Refering to the diagram in Case 2, we see the result of overflowing the last element of V'_t and the last element of V_t does not violate (a) and still preserves the condition that the last element of V'_t is not in V_t
- Suppose V'_t and V_t are in the same reverse ordering. Then overflowing the last elements of V'_t and V_t leaves V'_t and V_t still in the same reverse ordering. (a) is not violated since the same page is overflowed from M'^j and M^j.

Q.E.D.

PROOF of THEOREM 8

$M'^i \supseteq M^i$ for the same reasons as those used in THEOREM 7.

From LEMMA 8.4 $M'^j \supseteq M^j$.

Hence, $(M^i_t \cup (M^j_t)^i) \subseteq (M'^i_t \cup (M'^j_t)^i)$

Q.E.D.

4. Conclusions

We have developed a model of a data storage hierarchy system specifically designed for very large databases. This data storage hierarchy makes use of different page sizes across storage levels and maintains multiple copies of the same information in the hierarchy.

Four algorithms obtained from natural extensions to the LRU algorithm are studied in detail and key properties of these algorithms that affect performance and reliability of the data storage hierarchy are derived.

It is found that for the LOCAL-LRU algorithms, no choice of sizes for the storage levels can guarantee that a lower storage level always contains all the information in the higher storage levels. For the GLOBAL-LRU algorithms, by choosing appropriate sizes for the storage levels, we can (1) ensure the above inclusion property to hold at all times, (2) guarantee that no extra page references to lower storage levels are generated as a result of handling overflows, and (3) guarantee that no multi-level paging anomaly can exist.

Several areas of further study emerge from this investigation. These include the study of store-behind algorithms [18] and the study of extensions to other known storage management algorithms. We hope that this study motivates further work in the area of generalized data storage hierarchy systems for very large databases.

Acknowledgment

The authors would like to thank Mike Abraham, Sid Huff, and Ken Yip for reviewing an earlier version of this paper; and the referees for their editorial comments.

References

1. Arora, S.R., and Gallo, A. Optimal sizing loading and reloading in a multi-level memory hierarchy system. Proc. AFIPS 1971 SJCC 38, 337-344.

2. Belady, L.A. A study of replacement algorithms for a virtual-storage computer. IBM Systems Journal 5, 2 (1966), 78-101.

3. Belady, L.A., Nelson, R.A., and Shedler, G.S. An anomaly in space-time characteristics of certain programs running in a paging machine. Comm. ACM 12, 6 (June 1969), 349-353.

4. Chen, P.P. Optimal file allocation in multi-level storage systems. Proc. AFIPS 1973 NCC, 277-282.

5. Conti, C.J. Concepts for buffer storage. IEEE Computer Group News, March 1969, 6-13.

6. Denning, P.J. Virtual memory. ACM Computing Surveys 2, 3 (November 1974), 153-190.

7. Franaszek, P.A., and Bennett, B.T. Adaptive variation of the transfer unit in a storage hierarchy. IBM Journal of Research and Development 22, 4 (March 1978), 405-412.

8. Franklin, M.A., Graham, G.S., and Gupta, R.K. Anomalies with variable partition paging algorithms. Comm. ACM 21, 3 (March 1978), 232-236.

9. Greenberg, B.S., and Webber, S.H. MULTICS multilevel paging hierarchy. IEEE INTERCON 75.

10. Hatfield, D.J. Experiments on page size, program access patterns, and virtual memory. IBM Journal of Research and Development 16, 1 (January 1972), 58-66.

11. Hatfield, D.J., and Gerald, J. Program restructuring for virtual memory. IBM Systems Journal 10, 3 (1971), 168-192.

12. Hsiao, D.K., and Madnick, S.E. Data base machine architecture in the context of information technology evolution. Proc. Very Large Data Base Conf., Tokyo, Japan, Oct. 1977, 63-84.

13. Johnson, C. IBM 3850 - mass storage system. IEEE INTERCON, 1975.

14. Johnson, J. Program restructuring for virtual memory systems. MIT Project MAC TR-148 (March 1975).

15. Lam, C.Y., and Madnick, S.E. INFOPLEX data base computer architecture - concepts and directions. MIT Sloan School of Management Working Paper No. 1046-79 (1979).

16. Lum, V.Y., Senko, M.E., Wong, C.P., and Ling, H. A cost oriented algorithm for data set allocation in storage hierarchies. Comm ACM 18, 6 (June 1975), 318-322.

17. Madnick, S.E. Storage hierarchy systems. MIT Project MAC TR-105 (1973).

18. Madnick, S.E. INFOPLEX - hierarchical decomposition of a large information management system using a microprocessor complex. Proc. NCC 44 (May 1975), 581-587.

19. Madnick, S.E. Design of a general hierarchical storage system. IEEE INTERCON 75.

20. Madnick, S.E. The INFOPLEX database computer : concepts and directions. Proc. IEEE Computer Conference, February 26, 1979, 168-176.

21. Mattson, R.L., Gecsei, J., Slutz, D.R., and Traiger, I.L. Evaluation techniques for storage hierarchies. IBM Systems Journal 9, 2 (1970), 78-117.

22. Ramamoorthy, C.V., and Chandy, K.M. Optimization of memory hierarchies in multiprogrammed systems. Journal of the ACM 17, 3 (July 1970), 426-445.