AN ABSTRACT OF THE THESIS OF

Rudolph Joseph Frank (Name) for the Master of Science (Degree)

in Electrical and Electronics Engineering (Major) presented on 10-31-69 (Date)

Title: THE EFFECT OF BOTH HARDWARE AND SOFTWARE PARAMETERS ON THE EFFICIENCY OF A PAGING SYSTEM

Abstract approved: Redacted for Privacy

Professor Louis N. Stone

Due to the present trend to utilize both virtual memory and paging for the implementation of storage for large-scale computer systems, it is of interest to analyze the pertinent parameters associated with the operation of a paging system. This paper describes the paging process and determines the effect of variations in both hardware and software parameters on the efficiency of the system.

The software parameters of randomness, sequentialness, and recursiveness are introduced in the form of addressing patterns. From these patterns and a knowledge of the memory specifications of the system, an estimation of the average number of operations per page transfer is determined. The mean free path is also defined for the various addressing patterns.

The hardware parameter of instruction cycle time and the auxiliary storage parameters are introduced. From these values and the mean free path, an expression for processor ratio is developed. It is the variation of this processor ratio for different values of the hardware and software parameters which determines the effectiveness of the paging process. This variation is examined, and the interaction of the parameters on the performance of the system is determined thereby rendering a more effective paging system.

# The Effect of Both Hardware and Software Parameters on the Efficiency of a Paging System

by

Rudolph Joseph Frank

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

June 1970

APPROVED:

Redacted for Privacy

Professor of Department of Electrical and Electronics Engineering
in charge of major

Redacted for Privacy

Head of Department of Electrical and Electronics Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented 10-31-63

Typed by Barbara Eby for Rudolph Joseph Frank

## ACKNOWLEDGEMENT

It is the desire of the writer to thank Professor Louis N. Stone for his valuable help and comments in the preparation and writing of this thesis. The writer also wishes to thank the National Science Foundation for their support of this work. And lastly, the writer wishes to thank his wife, Donna, for her constant encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

# LIST OF TABLES

## LIST OF SYMBOLS

| Symbol | |
|---|---|
| A | Auxiliary storage size in words. |
| $A_i$ | Auxiliary storage allocated to program i. |
| B | Number of pages in auxiliary storage. |
| $B_i$ | Number of pages in auxiliary storage allocated to program i. |
| b | Block number associated with main memory address. |
| C | Core memory size in words. |
| $C_i$ | Core memory allocated to program i. |
| C(L, N) | Cycle set associated with recursive addressing pattern. |
| CPU | Central processing unit. |
| $E_i$ | Expected or average number of instructions per page transfer. |
| f | Probability that a page must be written out of core to make room. |
| $I_t$ | Instruction cycle time. |
| IR | Instruction register. |
| k | Number of requests in drum queue. |
| L | Cycle length of a recursive addressing pattern. |
| $l_m$ | Line number associated with main memory address. |
| $l_v$ | Line number associated with vertual memory address. |
| M | Main memory address. |

| | |
|---|---|
| MAR | Memory address register. |
| MFP | Mean free path. |
| $M_k$ | Deliverable paging rate from the drum system. |
| N | Cycle number or number of times cycle is repeated. |
| P | Page size in words. |
| PAR | Page address register. |
| $p_{rt}$ | Probability of needing a page transfer for a given operation in a random addressing pattern. |
| $p_{rn}$ | Probability of not needing a page transfer for a given operation in a random addressing pattern. |
| R | Number of pages in core. |
| $R_i$ | Number of pages in core associated with program i. |
| $R_p$ | Processing ratio or processor ratio. |
| S | Total storage in words. |
| SAR | Segment address register. |
| $S_i$ | Storage allocated to program i. |
| s | Segment number associated with virtual address. |
| T | Drum cycle time. |
| $T_p$ | Processing time. |
| $T_t$ | Total time. |
| U | Total number of pages in memory. |
| $U_i$ | Number of pages in memory associated with program i. |

| Symbol | |
|---|---|
| $V$ | Virtual address. |
| $\overline{W}$ | Average waiting time. |
| $X$ | Number of pages per track associated with drum. |
| $\tau$ | Transfer time or time to read a single page from drum. |

# THE EFFECT OF BOTH HARDWARE AND SOFTWARE PARAMETERS ON THE EFFICIENCY OF A PAGING SYSTEM

## I. INTRODUCTION

Virtual memory and paging are frequently utilized to economically implement large-capacity storage systems. In conventional paging systems the storage is separated into two distinct levels of hierarchy. The first level in main memory which is comprised of expensive fast-access core storage. The second level is auxiliary memory, which is cheaper than main memory but has a longer access time. To make the system economically practical, the cheaper auxiliary memory is used for the majority or bulk of storage. The system's efficiency therefore relies heavily upon two aspects:

1. The organization of the information within the distinct storage levels.
2. The manner in which the information is transferred between the two levels.

The transfer of information between the two levels is necessary because data in auxiliary memory is usually not directly available to the processor. Therefore any information in auxiliary memory, which is pertinent to the current program in operation, must be

removed from auxiliary memory and placed in main memory before it is available to the processor. In this way the fast access time associated with main memory may be utilized.

Since the swapping or transfer of information between auxiliary and main memory consumes expensive computer time, it is practical to swap information in groups of words rather than transfer words individually. These groups of words are called pages. The scheme or manner in which the groups are transferred is called paging. As stated in Flores (1967a), the interaction of data between the two levels may be managed by means of three alternatives:

1. The programmer assumes complete responsibility for the management of memory.
2. The computer system itself assumes the responsibility and the job is done automatically without the awareness of the programmer.
3. The programmer and the software cooperate.

The second alternative coincides with paging, and it will be examined since it is the most expedient and less likely for error.

The problem is to implement the system in an economical and efficient manner. Therefore the system's parameters and characteristics must be investigated. These parameters include page size, core size, total storage, instruction cycle time, type of auxiliary storage, and program characteristics. In this paper the auxiliary

storage media will be drum memory, and the programming characteristics will be defined according to their addressing patterns.

The purpose of this study is to analyze, for different degrees of sophistication in programming, the effect of both hardware and software parameters on the efficiency of the paging system.

# II. SEGMENTATION, AUXILIARY STORAGE, VIRTUAL MEMORY AND PAGING

Due to the cost of fast-access core storage, early computer systems were limited in the amount of core memory directly available to the user. Designers of systems with large storage requirements had but two alternatives with which to solve the problem:

1. Force programmers to divide their programs into segments, so that each segment could be loaded and processed individually until the job was completed.
2. Extend memory by introducing a cheaper auxiliary storage which is capable of storing the complete program within the domain of the computer memory.

## Segmentation

The success of the segmentation process is entirely dependent on the ingenuity of the programmer. In this situation the program storage requirement is greater than the amount of memory available. Once a program segment is completed, the next segment is loaded and processed to completion. The operation continues in this sequential manner until the entire program terminates.

The result of this segmentation process is a tremendous scheduling and housekeeping load on the programmer due to the following:

1. Deciding where to divide the program into appropriate segments.
2. Estimating the memory needed for each program segment; so that a memory overflow condition does not occur within a segment.
3. Assuring that all necessary variables, parameters, and constants are properly documented for the transition between segments.
4. Guarding against oversegmentation, which would increase overhead and decrease efficiency.

In addition to the above disadvantages of program segmentation, there is no guarantee that a program can be segmented efficiently given that the scheduling and housekeeping functions are performed properly. Certain programs simply require more core storage than is provided, and interior looping or radical branching would make the segmentation process impractical.

## Auxiliary Memory Under Program Control

The complexity of program segmentation can be relieved slightly by the utilization of auxiliary memory. Users can load their programs into main core until it is filled, and then empty the remainder of their program into auxiliary storage. Thus the complete program is within the domain of the computer memory.

The auxiliary memory can be randomly accessed as in the case of utilizing a slower core memory; cyclically accessed if disk or drum units are used; or serially accessed if magnetic tape units are used as the backing store.

The characteristics which all of the various types of auxiliary memories have in common are the following:

1. Auxiliary memory is cheaper per unit of stored information than main memory.
2. Auxiliary memory has a longer access time than main memory.
3. Auxiliary memory information is not directly available to the processor, and therefore must be transferred to main memory before it can be processed.
4. Auxiliary memory is larger than main memory.

A block diagram of a computer system utilizing auxiliary storage is illustrated in Figure 1. The output from auxiliary storage is the input into main core, and information in auxiliary storage cannot enter the central processor or the arithmetic unit. Certain system designs permit information transfer from auxiliary store to the output unit. The motive for this is an increase in the system performance due to the ability to simultaneously process and perform output operations utilizing the main core and auxiliary respectively.

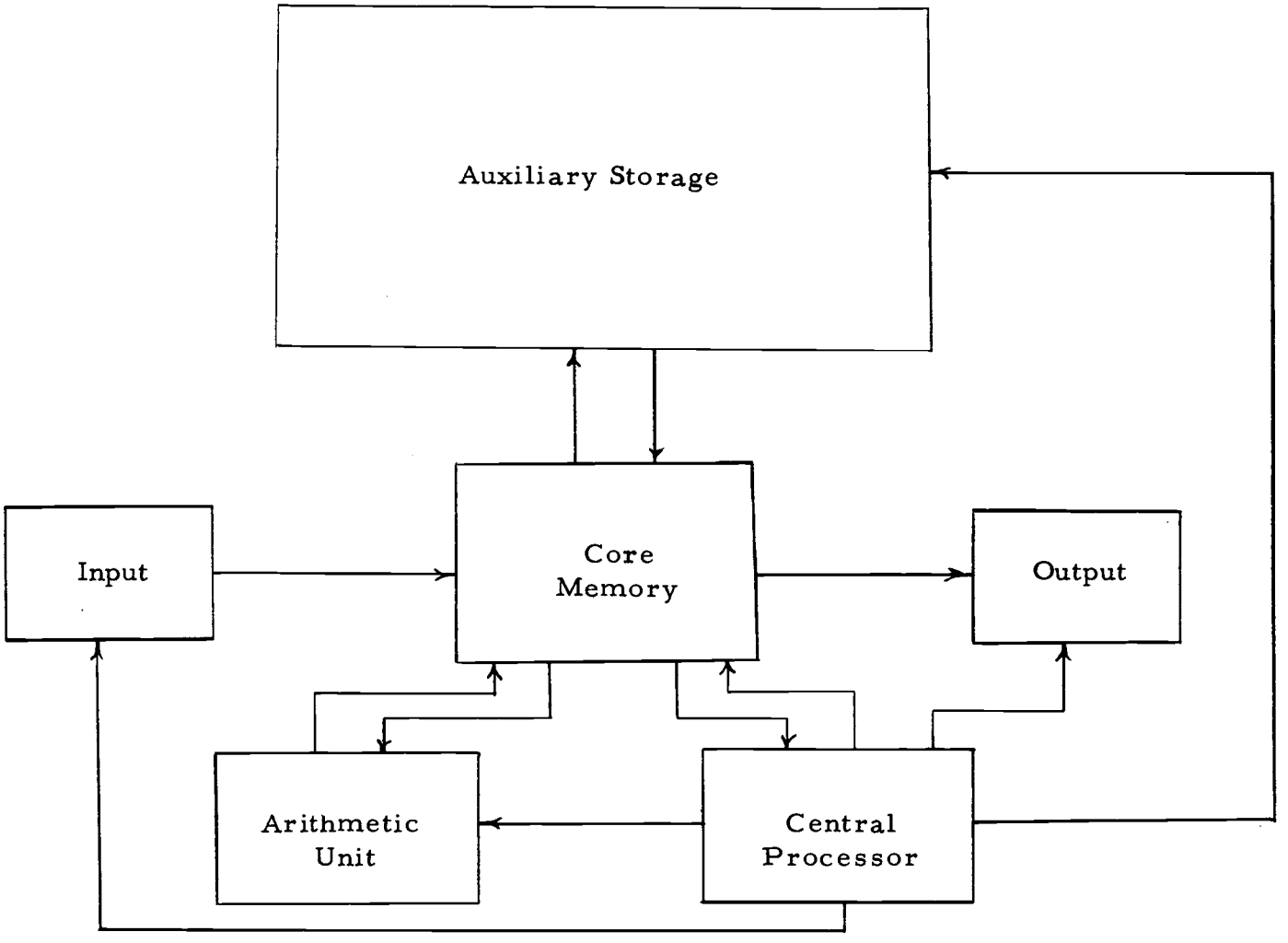



Figure 1. A computer system utilizing auxiliary storage.

The utilization of auxiliary memory considerably lessons the load on the programmer as compared with the segmentation scheme previously discussed. This stems principally from the fact that the complete program is always contained in the domain of the computer memory (either main core or auxiliary storage). The problems of memory overflow and the preparation of overlays for segment transition are avoided, but many of the other housekeeping problems are still present.

To illustrate this point, assume that a system has a main memory of fast-access core storage and one level of auxiliary memory, which is cyclically accessed drum storage. The entire program is loaded into core. Once the main memory is filled, additional information is forced onto the drum. Assuming the transfer of information is under program control, the programmer must specify what information in main memory is to be transferred to the drum. This is necessary in order to make room in main memory for new information generated by the program. The system processes and shifts information from core to drum until data which is residing in auxiliary memory is needed for processing.

At this point the programmer must know exactly where the needed information is located on the drum. Once located, the information is transferred back to main memory, which necessitates an additional transfer in order to provide space in main memory. Along

with each transfer the programmer must update his reference tables in order to keep an accurate record as to the location of the information. The sequence of processing, transferring, and updating continues until the termination of the program.

The housekeeping time associated with swapping information under program control is so complex, it could easily outweigh the processing time; not to mention the more severe problem of program error.

## Atomatic Utilization of Auxiliary Storage

The preceding sections have implied that neither program segmentation nor the utilization of auxiliary storage under program control are acceptable solutions to the problem of efficient processing in a computer system whose core memory capacity is smaller than the storage required by its programs. The pitfalls of the former methods rests in their dependency upon the programmer for success.

In order to liberate the programmer from the housekeeping burdens associated with transferring information, the system itself can be designed to handle the task. The objective is to make the total memory (main and auxiliary) appear to be at the disposal of the programmer. Thus the programmer and his program are independent of the level of storage which is utilized, and the responsibility for information transfer is placed upon the system.

The system must be capable of selecting in an efficient manner the information residing in main memory which must be transferred to auxiliary storage. Once the transfer has taken place the reference tables or registers must be updated. All these tasks are performed automatically and in such a way that system overhead does not predominate over system processing.

## Virtual Memory

A set of memory addresses which appears to be directly available to the programmer but actually is indirectly available under system control of auxiliary storage is termed a virtual memory. Flores (1967 a, b), Kilburn et al. (1962), Lauer (1967), and Fikes, Lauer, and Vareha (1968) have studied virtual memory through both analysis and simulation.

Much of the early computer literature attributes to the Atlas computer system of Manchester University, England, the distinction of being the first system to utilize virtual memory.

## Paging

Having conceded the need for automatic transfer of information between main and auxiliary storage, the next inquiry concerns the feasibility of transferring groups of words as compared with transferring words individually (Gibson, 1967). A typical CPU reference

to memory calls for a single word, maybe an instruction or a piece of data. Group transfer therefore provides more words than requested by a CPU reference. Often however the next reference is for an adjacent word or one near the previous virtual address. Since the new information of the latter reference may have already been transferred into main memory with the word group associated with the prior reference, there is a possibility of reducing the number of information transfers. This reduction of transfers is desirable since it is paralled by an increase in system efficiency.

A group of memory words under a designed label such that the entire group may be referenced by the label is termed a page. The process of transferring pages between main and auxiliary storage is called paging. The strategy or mechanism which the system utilizes to perform the paging process is called the paging scheme.

## Paging Scheme

An operational block diagram and a flow diagram of a typical paging scheme is illustrated in Figures 2 and 3 respectively. A page in main memory will be referred to as a block; and a page in auxiliary storage will be referred to as a segment. Both blocks and segments are fixed sections of their respective storage levels. A page address register (PAR) contains the labels of the pages residing in core, with a one to one correspondence between each page label

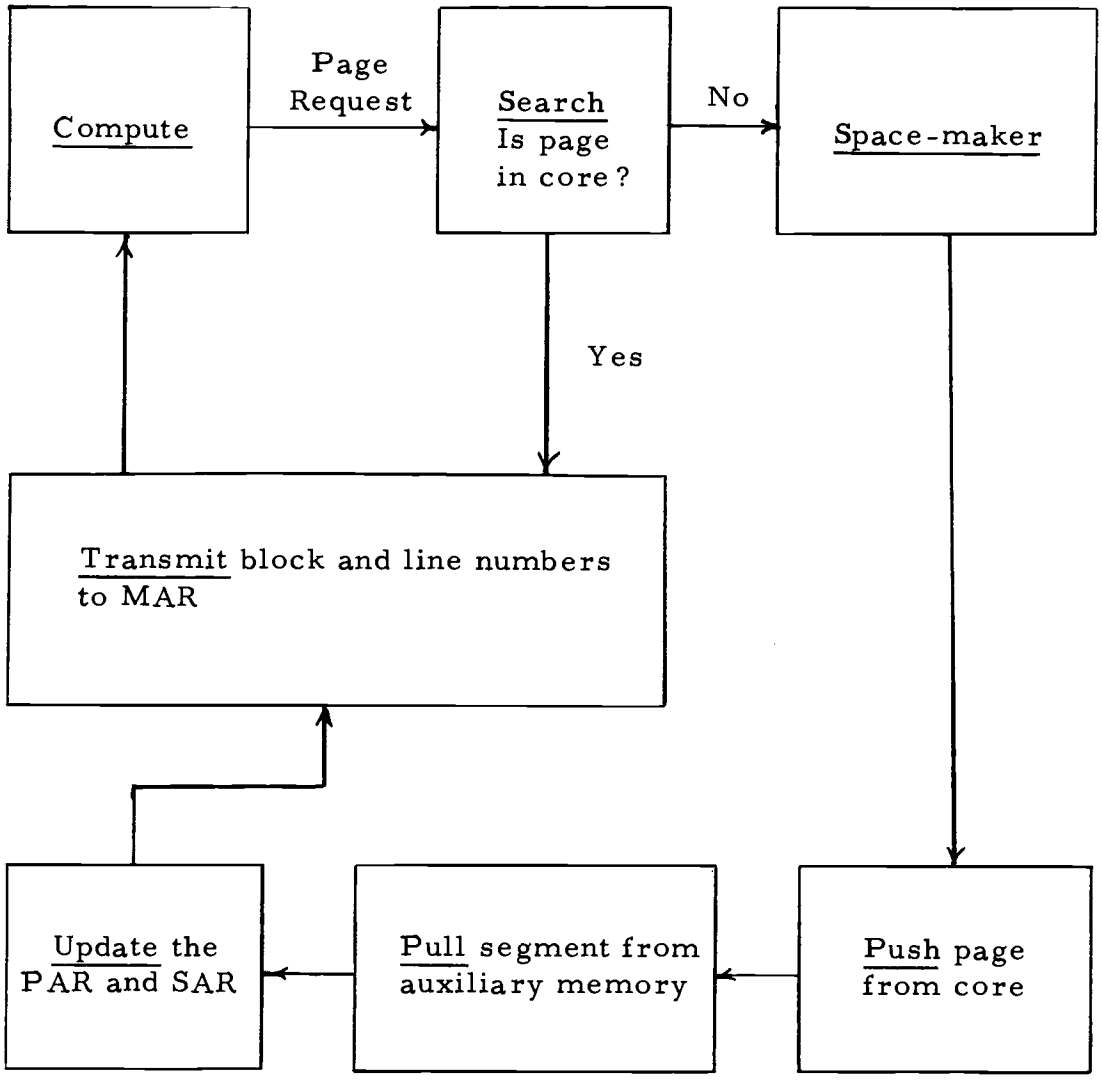



Figure 2. An operational block diagram of a paging scheme.

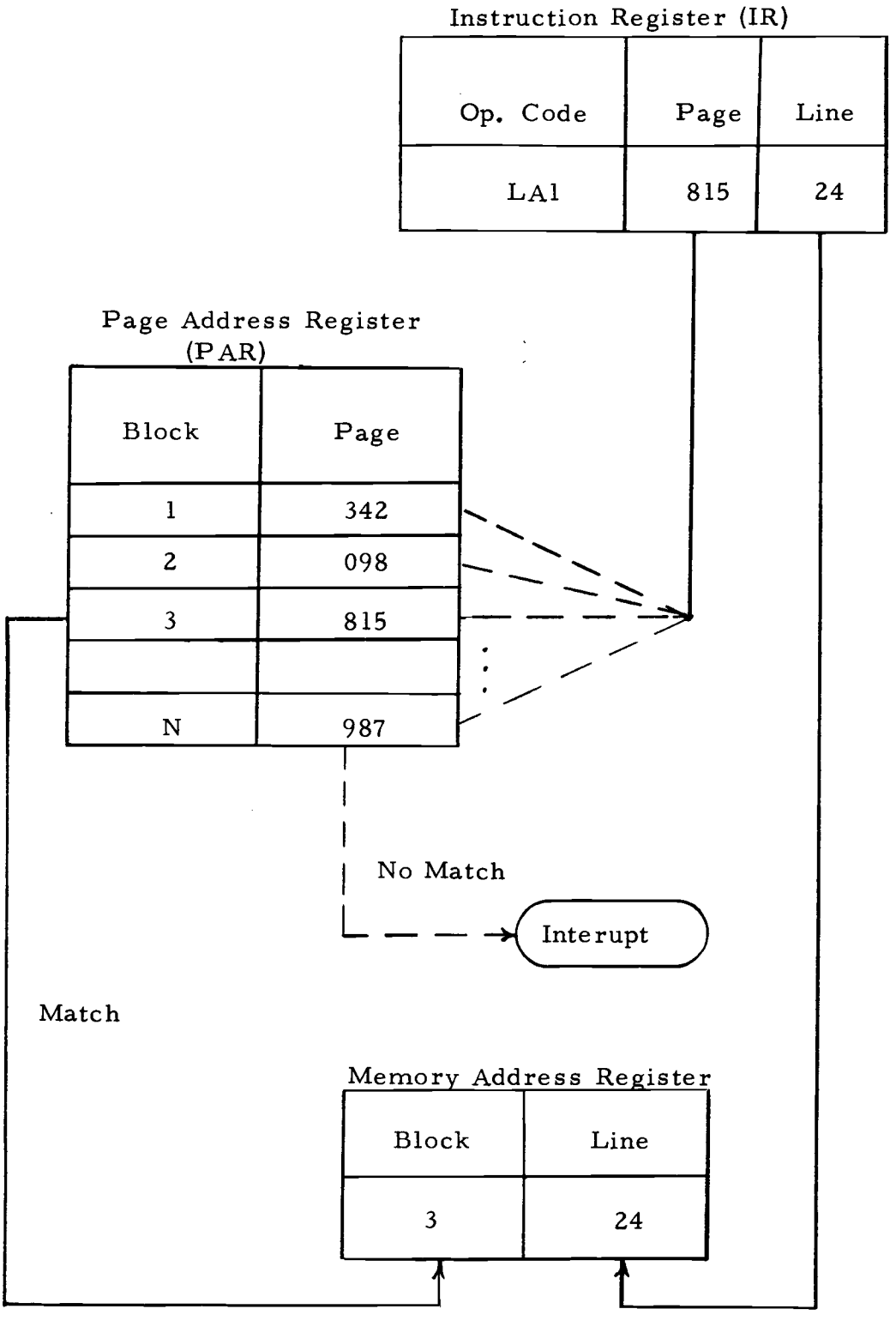



Figure 3. A. Flow diagram of paging scheme without page transfer.

and its associated core block. The segment address register (SAR) performs the same task with respect to auxiliary storage segments.

Let a virtual address V, denoted by $V = s, l_v$, be composed of a segment number s and a line number $l_v$. Likewise let a main memory address M, denoted by $M = b, l_m$, be composed of a block number b and a line number $l_m$. Assume the system is processing and the instruction register (IR) contains the operation code, page label, and the line number for the next instruction. Before this instruction can be processed, a compare operation is needed to determine if the page requested is residing in core memory. The compare operation can be simply a search of the page address register. If the IR page label matches one contained in the PAR, the page is in core and the block and line numbers are sent to the memory address register (MAR). The MAR points to the desired information in main core. If there is no match, the information is not in core and the interupt signal is activated.

During the interupt the virtual memory monitor is called to initiate a page transfer between main and auxiliary storage. The task of the monitor is to control a cycle of spacemaking, pushing, pulling and updating. This cycle is shown in Figure 3B.

The objective of the spacemaker is to choose a suitable page from a block in core to be transferred to an auxiliary storage segment in order to make room for the requested page in core. The idea is to select a page which will not be needed immediately; or more preferably the page least likely to be referenced in the near

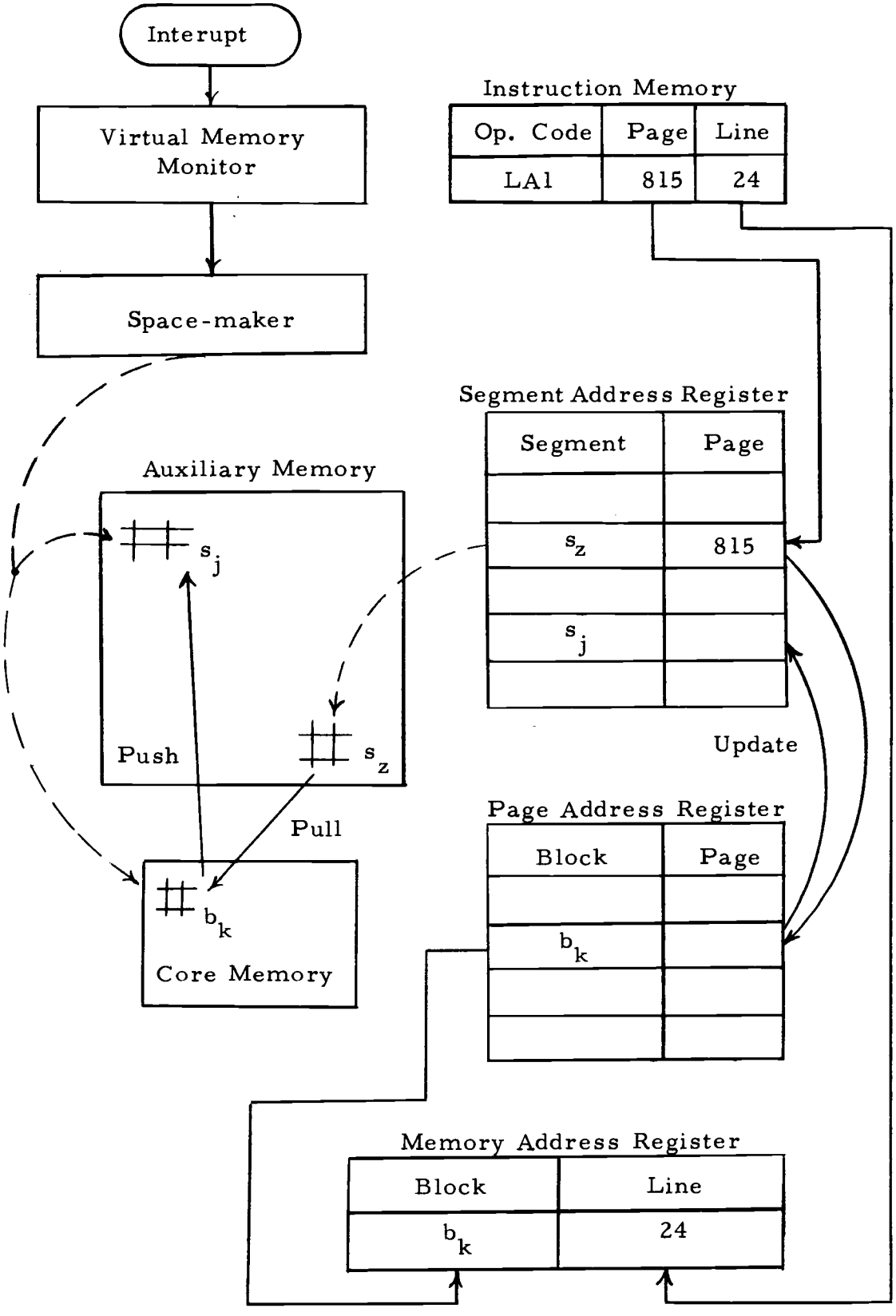



Figure 3.B. Flow diagram of paging scheme with page transfer.

future. Various designs ranging from random selection to elaborate statistical calculations have been proposed. (Belady, 1966; Belady, Nelson and Shedler, 1969). In certain cases a simple but fast spacemaker may be more desirable than a sophisticated, near optimal one. In any event the design of a spacemaker is a compromise between sophistication and speed. However, it should be recognized that their performance is pertinent to the efficiency of the paging system.

A push is a transfer of a page from a block in core memory to a segment in auxiliary memory. The push operation follows the decision of the spacemaker. A pull is a page transfer from auxiliary storage to core. It is advantageous to have block and segment sizes equal for both the push and pull operations. The time involved in performing these operations has an adverse effect on the efficiency of the system. The speed of the push and pull transfers depend on the following two factors:

1. The access time of the auxiliary storage device utilized.
2. The information transfer rate between the two storage levels.

Following the pull operation the PAR and SAR are updated and the block and line numbers are placed in the MAR, this completes the cycle controlled by the virtual memory monitor. At this point the system has the requested page in core memory and is capable of processing. The total cycle is again initiated upon the next page request from the instruction register.

It should be emphasized that the previous paging scheme was just one possible method of implementing the paging process. No attempt has been made here to optimize its performance. The illustration was made principally to point out the individual operations which are performed during the paging cycle. Attempts have been made to optimize existing systems by Wallace and Mason (1969), and McKellar and Coffman (1969).

## Time-sharing

The concept of separating storage into word groups under a specific level makes the principle of paging attractive when designing a time-sharing system. The reason is that a time-sharing system presupposes a fragmentation of storage among the users, and paging facilitates the implementation of fragmentation. Each user's program may be assigned specific blocks in main core and specific segments in auxiliary storage. When a particular program requires a page transfer or processes for the duration of its time slice, the next program is serviced under the control of the executive.

The order of servicing can be round-robin, strict priority, or first come-first serve. Regardless of the method of ordering each program is required to wait while another is processing. And this is where time-sharing and paging complement one another, since the swapping of pages can take place during the waiting time. Therefore

the efficiency of the paging mechanism is higher in a multi-user mode than in a single user mode.

Because of either the arrangement of information on the auxiliary storage device or the order of servicing the various tasks, the situation may arise where all the programs are in the wait status. When this occurs the CPU is idle and the entire system is held in an inactive state until a page transfer is completed. Many present-day paging systems have been shown to be ineffectual because of the frequent occurance of CPU idleness (Lauer, 1967).

# III. PAGING PARAMETERS AND ADDRESSING PATTERNS

In the preceding chapter the motives and fundamental concepts associated with paging were discussed. From these concepts it can be inferred that the efficiency of the system is dependent on the relative number of page transfers which must be performed per unit of operating time. In order to estimate the rate of page transfers, the system itself has certain parameters which must be determined. These parameters include page size, core memory size, auxiliary storage size, instruction cycle time, and the page transfer rate.

To complement these system parameters there are certain software characteristics which influence the efficiency of the system. These characteristics are the addressing patterns of a given repertoire of programs. The addressing patterns may be random, sequentail, or recursive.

In this chapter the various system parameters and addressing patterns will be defined, and their mutual influence on the rate of page transfers will be examined.

## Storage Notation

The total storage (S) of a computer system is the sum of the main core capacity in words plus the auxiliary storage capacity in words. In general, the total storage may be expressed by the equation

$$S = \sum_{i=1}^{n} S_i \ ,$$

where $n$ is the total number of programs sharing storage,

$i$ is the designation of a particular program, and

$S_i$ is the amount of storage allocated to program $i$.

Let $C$ designate the core memory size in words. If $C_i$ equals the core memory associated with program $i$, then

$$C = \sum_{i=1}^{n} C_i \ .$$

Let $A$ designate the auxiliary storage size in words. If $A_i$ equals the auxiliary storage associated with program $i$, then

$$A = \sum_{i=1}^{n} A_i \ .$$

Let the page size in words be denoted by $P$, and assume the block and segment sizes are both equal to the page size. If $R$ designates the number of pages contained in core, then

$$R = C/P \ .$$

If $R_i$ designates the number of pages in core allocated to program $i$, then $R_i = C_i/P$. Again we may write

$$R = \sum_{i=1}^{n} R_i \ .$$

Let B designate the total number of pages in auxiliary storage, then $B = A/P$. If $B_i$ denotes the number of pages in auxiliary storage associated with program i, then $B_i = A_i/P$. In general,

$$B = \sum_{i=1}^{n} B_i \ .$$

Let U designate the total number of pages in core and auxiliary storage combined. Therefore $U = S/P$. If $U_i$ denotes the total number of combined pages associated with program i, then $U_i = R_i + B_i$, and

$$U = \sum_{i=1}^{n} U_i \ .$$

## Random Addressing Patterns

Definition 3.1. A random addressing pattern is a sequence of CPU references to memory where at any point in the addressing sequence any address is equally likely to be referenced.

Since in a random addressing pattern there is an equal probability of referencing any word contained in the storage domain of a particular program, the probability $p_{rt}$ of needing a page transfer for a given operation is equal to the ratio of auxiliary storage to total

storage for that particular program. Therefore $p_{rt} = A_i/S_i$. In the case of a uniprogrammed system $p_{rt}$ is simply the ratio of auxiliary storage to total storage. The interesting fact concerning random addressing patterns is that the probability of a page transfer is independent of the page size. Therefore in the random case the theory of transferring groups of words, as compared with transferring words individually, does not yield a significant reduction in information swapping between the two storage levels. This may be an indication that programs having random addressing patterns are not suited for a paged memory environment.

The probability of not needing a page transfer for a single reference in a random addressing pattern is denoted by $p_{rn}$. Therefore since the previous two events were mutually exclusive and exhaustive, we may write

$$p_{rn} = 1-p_{rt} = 1-A_i/S_i = (S_i - A_i)/S_i = C_i/S_i$$

The term $C_i/S_i$ is referred to as the core ratio for program i, since it is the ratio of core to total storage for that program. The two limiting cases occur when $p_{rn} = 0$ and $p_{rn} = 1$. In the former case the auxiliary storage dominates over core, and practically every reference requires a page transfer. In the latter case almost the entire memory is composed of core, and not many page transfers are necessary. Both of these limiting cases are impractical for a

paged system. Nevertheless some method of evaluating the effectiveness of core ratio between these two limits is necessary since most programs possess a certain degree of randomness.

To illustrate a possible method of evaluation, let an instruction cycle be composed of a operation plus a CPU reference to memory. Also assume the program is characterized by random addressing patterns. The probability of no page transfer on the first instruction cycle is $p_{rn}$, which we can write $p_{rn} = C_i/S_i = R_i/U_i$ for a fixed page size P. Also the probability of no page transfer on either the first or second instruction cycle is $(R_i/U_i)^2$. Likewise the probability of no page transfer on the first k cycles is $(R_i/U_i)^k$. Therefore the average or expected number of instruction cycles, $E_i$, before a page transfer is needed can be expressed by the equation

$$E_i = \sum_{k=0}^{\infty} k(R_i/U_i)^k \quad .$$

Figure 4 shows a plot of $E_i$ as a function of core ratio.

The results of Figure 4 indicate that random addressing patterns require a core ratio of 0.5 in order to average two instructions per page transfer. This means that with a core size equal to that of the auxiliary storage, the system can be expected to require a page transfer on every other instruction. With a core ratio of 0.8 the system requires a page transfer every 20 instructions, but this

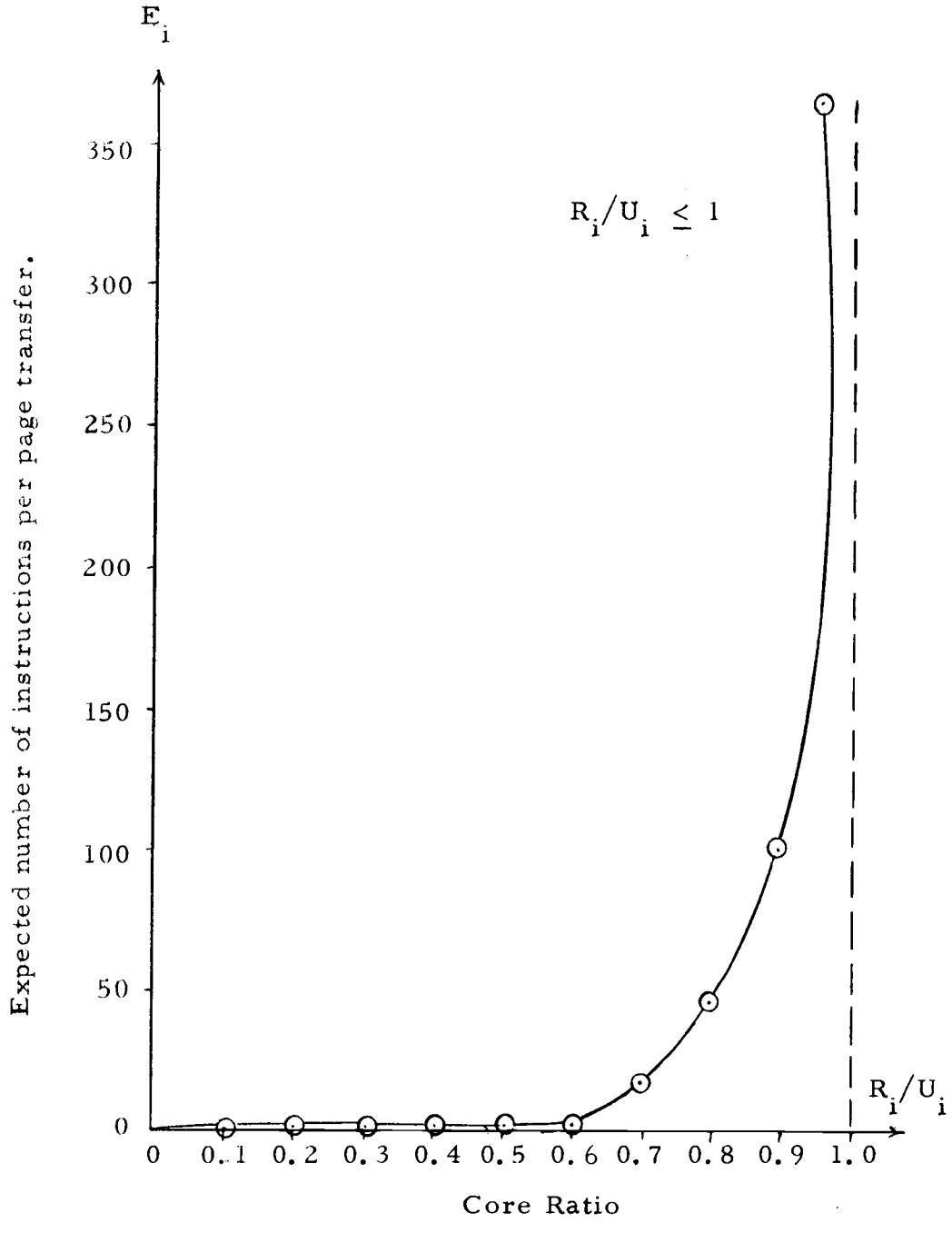



Figure 4. Expected number of instructions per page transfer as a function of core ratio for random addressing patterns.

implementation necessitates a core memory which is four times larger than the auxiliary storage. Therefore by recalling the assumption that auxiliary storage be larger than core, one can state that programs dominated by random addressing patterns are not practical for a paging environment.

## Sequential Addressing Patterns

Definition 3.2. A sequential addressing pattern is a sequence of CPU references to memory where at any point in the addressing sequence a particular address is exactly one word away from the preceding address.

As an example, let a CPU reference be composed of a page number and a line number. If the addressing pattern is strictly sequential, the next CPU reference will have the same page number with the line number incremented by one. When a current CPU reference contains the last line number of a page, the next reference will have the page number incremented by one and the line number will be reset to zero.

If a program is processing from a specific page in core, this page will continue to be referenced until the last word of the page is used. Once the entire page is referenced, it will not be referenced again since the addressing pattern is assumed to be purely sequential. Therefore, the program will start at its initial reference and process

through the pages resident in core until the core is exhausted. At this time a page transfer will be required.

The spacemaker pushes a page from core and a pull operation makes the next page in the sequence available. All the remaining pages in core which have previously been referenced will remain dormant in core for the remainder of the program. Since there are $P$ words on each page, the program is now limited to at most $P$ instructions per page transfer until the termination of the task.

Now suppose that an average program can fit within $U_i$ pages of storage and $R_i$ of these pages are within core. This means that a total of $R_i \times P = C_i$ words are available to be sequentially referenced before needing a page transfer. The new page will be used for $P$ references and terminate with another page transfer. The total number of references used by the program will be $U_i \times P = S_i$. Therefore the program will require $(S_i - C_i)/P$ page transfers for its completion. This means that the average number of operations per page transfer will be $\dfrac{S_i}{(S_i - C_i)/P}$.

Table 1 lists the number of page transfers needed to complete a program as a function of the page size for different values of the core ratio and program size. The following results are shown in Table 1:

1. As the page size is decreased, the number of page transfers

TABLE 1. Required number of page transfers to complete a program occupying $S_i$ words of storage for the sequential case.

| Page Size (words) | Required Number of Page Transfers to Complete Program Occupying $S_i$ Words of Storage | | |
|---|---|---|---|
| | $C_i/S_i = 1/4$ | $C_i/S_i = 1/8$ | $C_i/S_i = 1/16$ |
| | $S_i = 4096$ | | |
| 2 | 1536 | 1792 | 1920 |
| 4 | 768 | 896 | 960 |
| 8 | 384 | 448 | 480 |
| 16 | 192 | 224 | 240 |
| 32 | 96 | 112 | 120 |
| 64 | 48 | 56 | 60 |
| 128 | 24 | 28 | 30 |
| 256 | 12 | 14 | 15 |
| 512 | 6 | 7 | -- |
| 1024 | 3 | -- | -- |
| 2048 | -- | -- | -- |
| 4096 | -- | -- | -- |
| | $S_i = 8192$ | | |
| 2 | 3072 | 3584 | 3840 |
| 4 | 1536 | 1792 | 1920 |
| 8 | 768 | 896 | 960 |
| 16 | 384 | 448 | 480 |
| 32 | 192 | 224 | 240 |
| 64 | 96 | 112 | 120 |
| 128 | 48 | 56 | 60 |
| 256 | 24 | 28 | 30 |
| 512 | 12 | 14 | 15 |
| 1024 | 6 | 7 | -- |
| 2048 | 3 | -- | -- |
| 4096 | -- | -- | -- |
| | $S_i = 16,384$ | | |
| 2 | 6142 | 7168 | 7680 |
| 4 | 3072 | 3584 | 3840 |
| 8 | 1536 | 1792 | 1920 |
| 16 | 768 | 896 | 960 |
| 32 | 384 | 448 | 480 |
| 64 | 192 | 224 | 240 |
| 128 | 96 | 112 | 120 |
| 256 | 48 | 56 | 60 |
| 512 | 24 | 28 | 30 |
| 1024 | 12 | 14 | 15 |
| 2048 | 6 | 7 | -- |
| 4096 | 3 | -- | -- |

is increased proportionally.

2. As the core ratio is decreased, the number of page transfers is increased moderately.
3. As the program size is increased the number of page transfers is increased proportionally.

Figure 5 shows the influence of the same parameters on the average number of operations per page transfer. In each case the page size is the dominating parameter affecting the average. The core ratio determines how many references can be made before the initial page transfer and therefore tends to make the average greater than the page size. As the core ratio is increased, the average number of operations per page transfer will tend to surpass the page size by a greater amount.

The program size has little effect on the average since the program size is determined by the number of references, and doubling the number of references will also double the required number of page transfers. The results will be no change in the average number of operations per page transfer since both the number of operations and the number of page transfers has doubled.

The sequential addressing patterns are more attractive to a paging system than random patterns. This fact can be verified by comparing the respective values of $E_i$ for the same core ratio. For instance, in the random case with a core ratio of 0.25, the value of

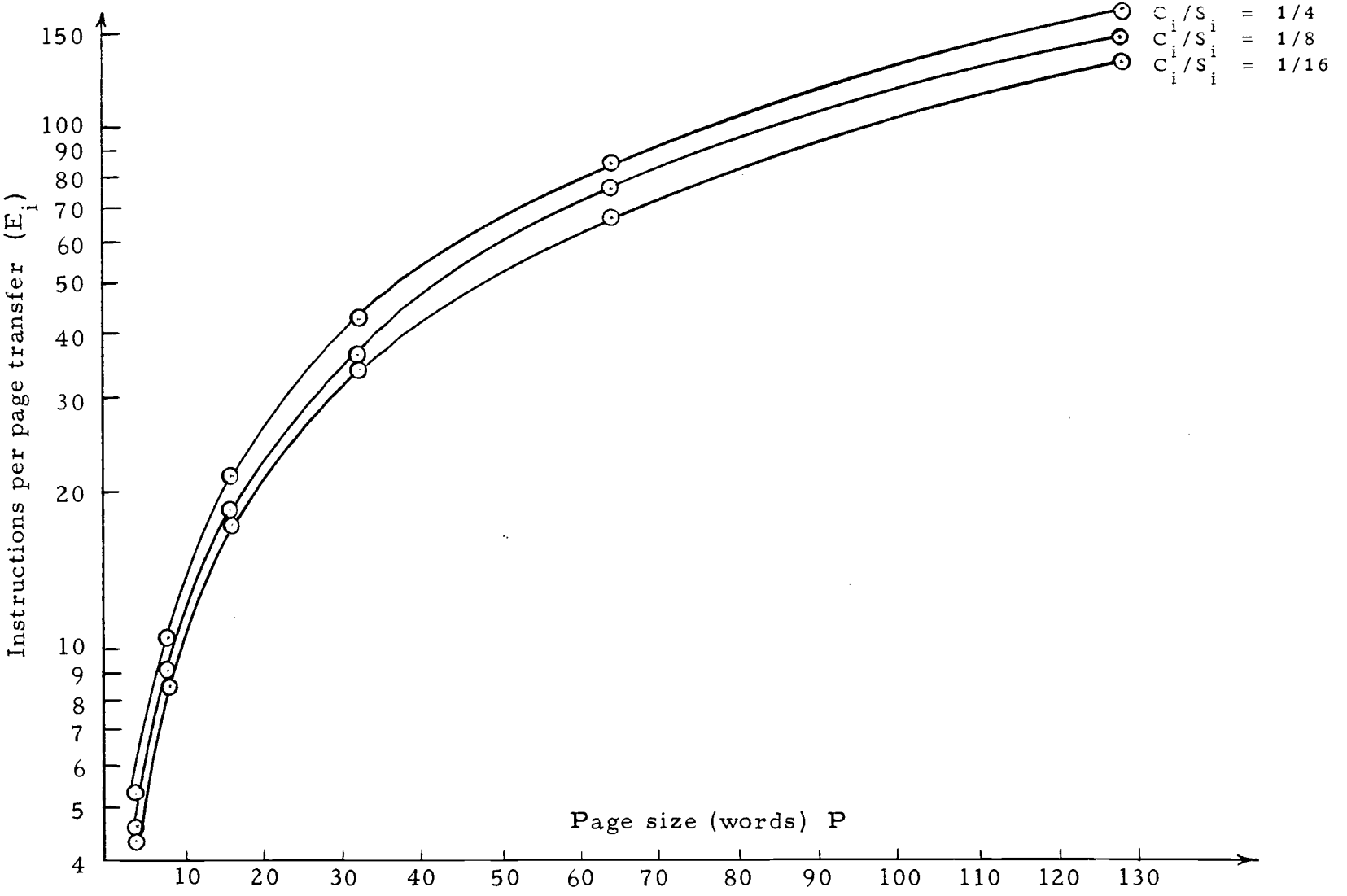



Figure 5. Expected number of instructions per page transfer as a function of the page size for sequential addressing patterns.

$E_i$ is less than 2 operations per page transfer. This means that nearly every other reference will require a page transfer. In the sequential case with the identical core ratio and with a page size of 4 words, the value of $E_i$ is greater than 5 instructions per page transfer. In the sequential case with a page size of 64 words, the value of $E_i$ is greater than 80 instructions per page transfer. Therefore the sequential patterns appear more attractive.

## Recursive Addressing Patterns

Definition 3.3. A recursive addressing pattern is a sequence of CPU reference to memory where the references in the sequence are repeated either in a cyclic or haphazard manner.

Recursive patterns appear mainly because of program loops, continual branching within a local segment of the program, or because of frequent references to subroutines. The most common application is in the case of matrices and matrix operations. (McKellar and Coffman, 1969).

To represent a cyclic-recursive pattern mathematically a cycle length (L) and a cycle number (N) should be established. The cycle length is the number of instructions contained in the cycle. The cycle number is simply the number of times the cycle is repeated. Therefore each cyclic pattern will be represented by a cycle set C(L, N). The cycle is said to terminate when another addressing

pattern is recognized.

In utilizing cyclic addressing patterns one must be careful of the occurance of page transfers within the cycle. This is especially important for small cycle lengths. Suppose the cycle number is 100 and the cycle length is 10 instructions with each instruction containing a CPU reference such that no two references are within the same page. If the program is alloted less than 10 pages, and the replacement algorithm is of the first in-first out type, there will be a page transfer on every reference following the first cycle. This continual swapping of pages will last for 100 cycles thereby causing nearly 1000 consecutive page transfers. The effect of this on system efficiency is disastrous.

The pitfalls of the preceding example rest on the fact that each reference was residing in a different page, and the number of pages associated with the program was less than the cycle length. If any one of these difficulties had not occurred, the program might have run consecutively through 1000 instructions without a page transfer.

In a haphazard-recursive pattern the requirement is merely that the CPU references be limited to a neighborhood of pages, and that the program function for a given number R of instructions without leaving the neighborhood. The difference between the cyclic and haphazard patterns is that the cyclic patterns emphasize a

repeated order, while the haphazard patterns emphasize a more random order confined to a local neighborhood of memory. Problems in utilizing haphazard patterns will occur if the neighborhood is larger than the core allocated to the program.

## Patterns and Paging

It was previously stated that sequential patterns are more attractive to a paging environment than random patterns. Now considering that recursive patterns can be associated with a given number of operations within a local neighborhood of memory, a similar comparison between sequential and recursive patterns can be made by considering the mean free path.

Definition 3.4. The mean free path is the average time duration between consecutive page transfers. The mean free path is denoted by MFP, and can be expressed by the equation $MFP = E_i \times I_t$, where $E_i$ and $I_t$ are the instructions per page transfer and instruction cycle time respectively.

Assume that the desired MFP for a paging system is set at one second, and an instruction cycle takes one microsecond for completion. This means that approximately one million CPU references to memory must be completed in order to process for one second without interruption. For a sequential addressing pattern this implies that core memory contain one million words so that the

references may be made. However for a recursive pattern the demand for core space is not nearly as large due to the fact that many of the references are repeated. Therefore because recursive patterns require less storage for a given mean free path, and because recursive patterns are not limited by the system in the number of operations per page transfer; it appears that recursive patterns are more attractive to a paging environment than sequential patterns.

Although the three types of addressing patterns were treated separately and effected the system differently, there is no requirement that a single program cannot possess several different patterns. If a program sequentially operates on a set of variables, then recursively performs numerous loop commands, and finally branches radically from one location to another; the only possible statement concerning the addressing pattern of this program is that it contains varying degrees of randomness, sequentialness, and recursiveness.

# IV. THE EFFECTS OF THE AUXILIARY STORAGE PARAMETERS ON THE PAGING SYSTEM

As mentioned in Chapter 2, there are various types of auxiliary memory units which may be utilized in a paging system. The slow access time characteristic of the magnetic tape units, and the expense associated with the random-access core memory has lead to the utilization of drum and disk as the common means of providing auxiliary storage.

Both drum and disk units are cyclically accessed and therefore possess rotational delays. The disk units also have additional translational delays. For the sake of simplicity, the drum memory will therefore be analysed as a media of auxiliary storage.

## The Drum System

The drum system which is to be considered is of the same configuration as appears in Lauer (1967). The drum operates asynchronously from the CPU, and under the control of its own channel program. A new channel program is normally initiated at the start of each drum revolution. Its time duration is equal to the drum cycle time (T). All page requests must be received and analyzed prior to the beginning of a channel program if they are to be transferred during its duration. All requests not meeting this

criterion are held over for the next channel program.

The circumference of the drum is divided into ring shaped surfaces called tracks. Each track is equipped with one read-write head and is restricted in the sense that only one head may be connected to the drum channel at any time. On the surface of each track there is enough memory to record X pages, and there is also ample space to permit head switching between consecutive pages. Therefore the first head, which may be permanently connected to trach a, can be connected to the channel program and the first page is read from track a . Next while the drum is turning between pages, the channel may be switched to the fourth head which is connected to track d . The second page is then read on track d . This order continues unith the $X^{th}$ page is read on one of the tracks. From a system's point of view, it appears that there is only one head with X slots passing under the head. The operating conditions are that only one page may be written in each slot and only at the instant the slot is under the head. A diagram of the drum system is shown in Figure 6.

To illustrate the utilization of the drum memory, assume that three programs are sharing memory simultaneously and each program executes in order until it requests a page from the drum. As seen in Figure 7, which shows the utilization of the drum system, program 1 processes until a page is requested at time B. Beginning at time B, programs 2 and 3 process in order until pages are

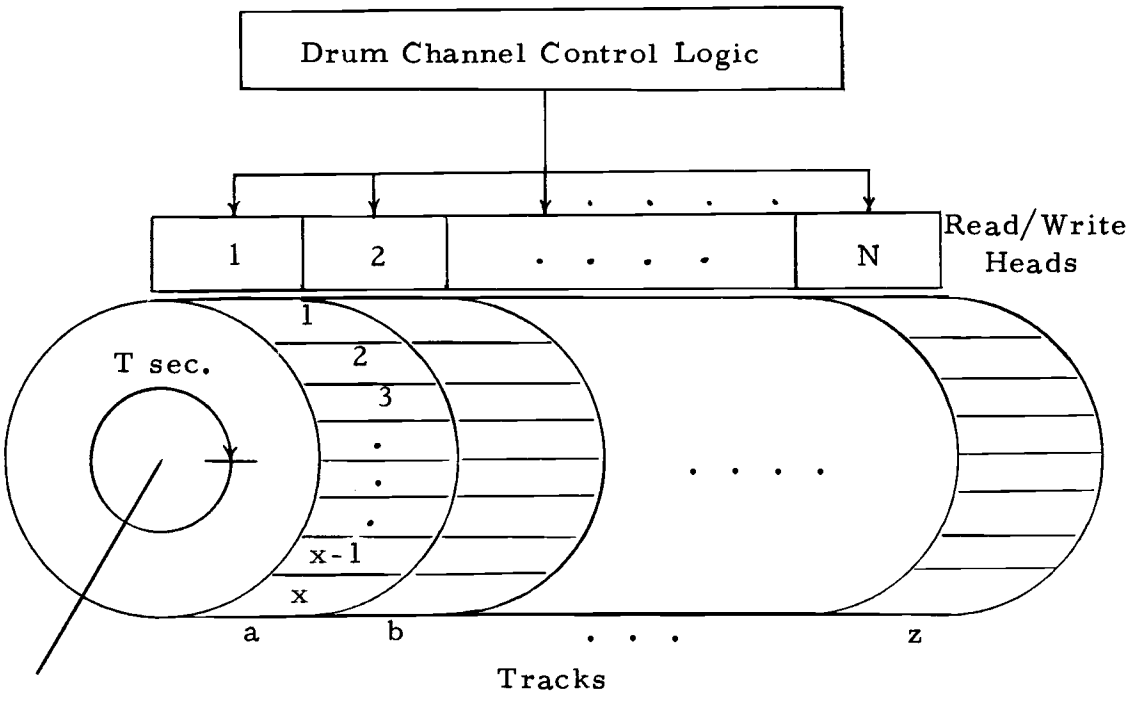
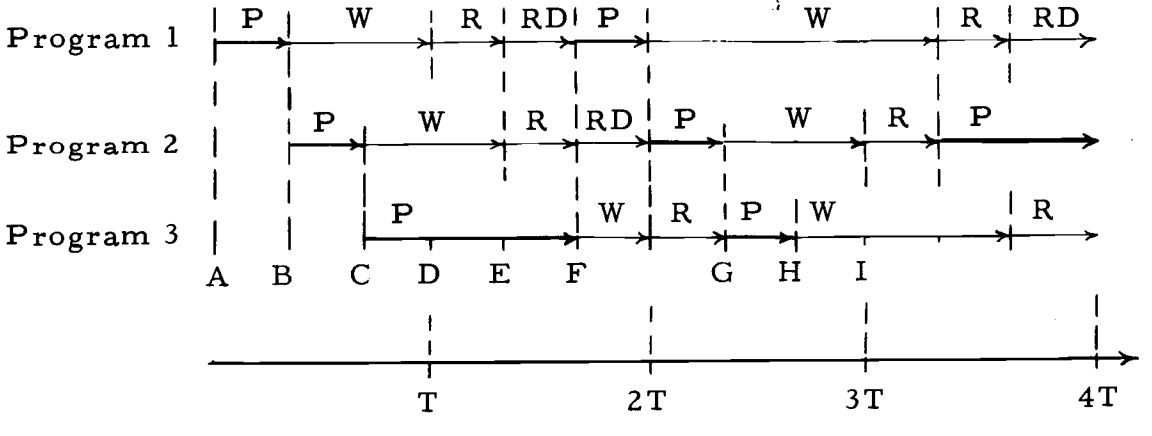



Figure 6. The paged drum system.




P = processing R = reading

W = waiting RD = ready

Figure 7. Utilization of the drum system.

requested at times C and F respectively. Simultaneously the drum channel program is running for a complete drum revolution of time T. At time D the page request of program 1 is located and read into core until time E. Program 1 therefore remains in the ready state until time F. Thus far the system has complete processing and swapping overlap.

At time G all programs enter the wait state because of a slot conflict. This situation occurred because program 1 requested a page in the same slot as the prior request of program 3. Therefore the system must wait until time I, when program 2 is ready, before processing can continue. The result is that all the programs were required to wait from time G to I while the drum was rotating. From the viewpoint of the system the effects of the rotational delay can be seen in the following observations:

1. The CPU is held in an idle state.
2. The processing and swapping overlap is terminated.
3. The users feel the effect in a poor response time.

From these observations one can anticipate the need to minimize the waiting time and devise a means of measuring its effect upon the system.

## Processing Ratio

Definition 4.1. The processing ratio ($R_p$) of a particular task is the ratio of the processing time ($T_p$) to the total time ($T_t$) associated with the task.

The total time is the sum of the processing time and the average waiting time ($\overline{W}$) of the task. Therefore the processing ratio can be expressed by the following equation:

$$R_p = \frac{T_p}{T_p + \overline{W}} \tag{4.1}$$

Lauer (1967) showed that the average waiting time which a task spends waiting for a single page is a function of the rotational delay of the drum. This waiting function is given below and its development appears in Appendix A. Therefore,

$$\overline{W} = T(1 + \frac{1}{2X}) \ , \tag{4.2}$$

where $T$ is the cycle time of the drum, and $X$ is the number of pages per track.

The processing time may be approximated by the equation,

$$T_p = E_i \times I_t \tag{4.3}$$

where $E_i$ is the average number of instructions per page transfer, and $I_t$ is the instruction cycle time.

Equations 4.2 and 4.3 can be substituted into Equation 4.1 yielding

$$R_p = \frac{E_i \times I_t}{(E_i \times I_t) + T \times (1 + \frac{1}{2X})} \qquad (4.4)$$

## The Effect on the Processor Ratio of Varying the Drum Size While Holding the Transfer Time and Page Size Constant

The transfer time ($\tau$) is the amount of time required to read a single page from the drum. Since there are X slots positioned on a single track, the transfer time can be written

$$\tau = \frac{T}{X} .$$

If both T and X are varied such that the transfer time is held constant as shown in Figure 8, the effect on the processor ratio of

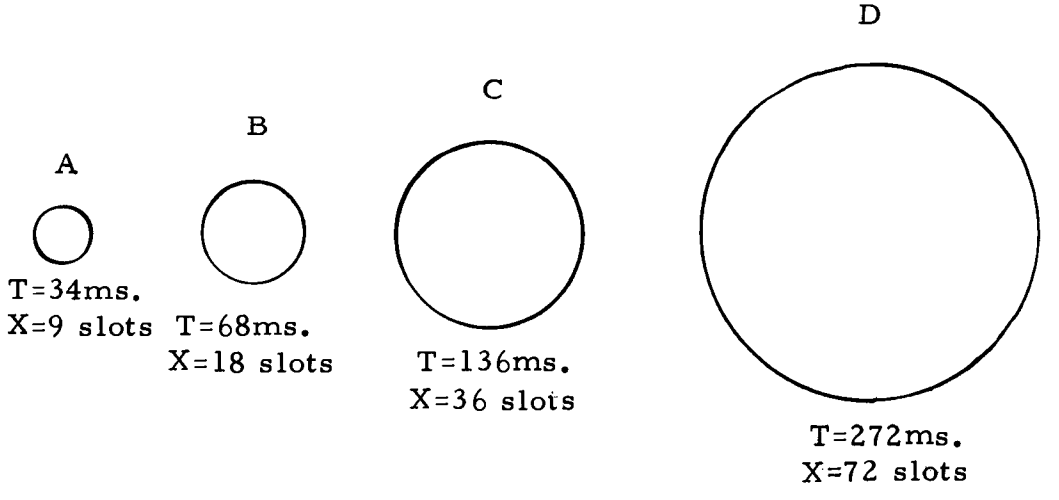



Figure 8. Four different drum sizes with the same transfer time $\tau = T/X = 3.7$ms.

varying the physical size of the drum may be determined. Figure 9 shows the effect of this variation for the four cases of Figure 8 with $\tau = 3.7$ ms. and $I_t = 10\,\mu s$.

Figure 9 shows plots of processor ratio as a function of instructions per page transfer for different size drum units. In all four cases the processor ratio increased as the number of instructions per page transfer increased. This result is logical because as the number of instructions per page transfer is increased, there is a parallel increase in the mean free path since the instruction cycle time was held constant. Therefore larger intervals between page transfers would account for the increase in processor ratio.

Another pertinent result is that the processor ratio increases as the size of the drum decreases. For small values of $E_i$ the physical size of the drum is not significant, this is shown in Figure 9 for values of $E_i$ less than 100. For larger values of $E_i$, a reduction in drum size by a factor of 2 may increase the processor ratio by 50%. For example, comparing curves C and B in Figure 9 at $E_i$ equal to 8000 instructions per page transfer will yield processor ratios of 0.367 and 0.534 respectively. Therefore by reducing the number of pages per track from 36 to 18, the processor ratio is increased 46%. At the same value of $E_i = 8000$, reducing the number of pages per track from 72 to 36 will yield an increase of 65% in the processor ratio. For extremely large values of $E_i$ the

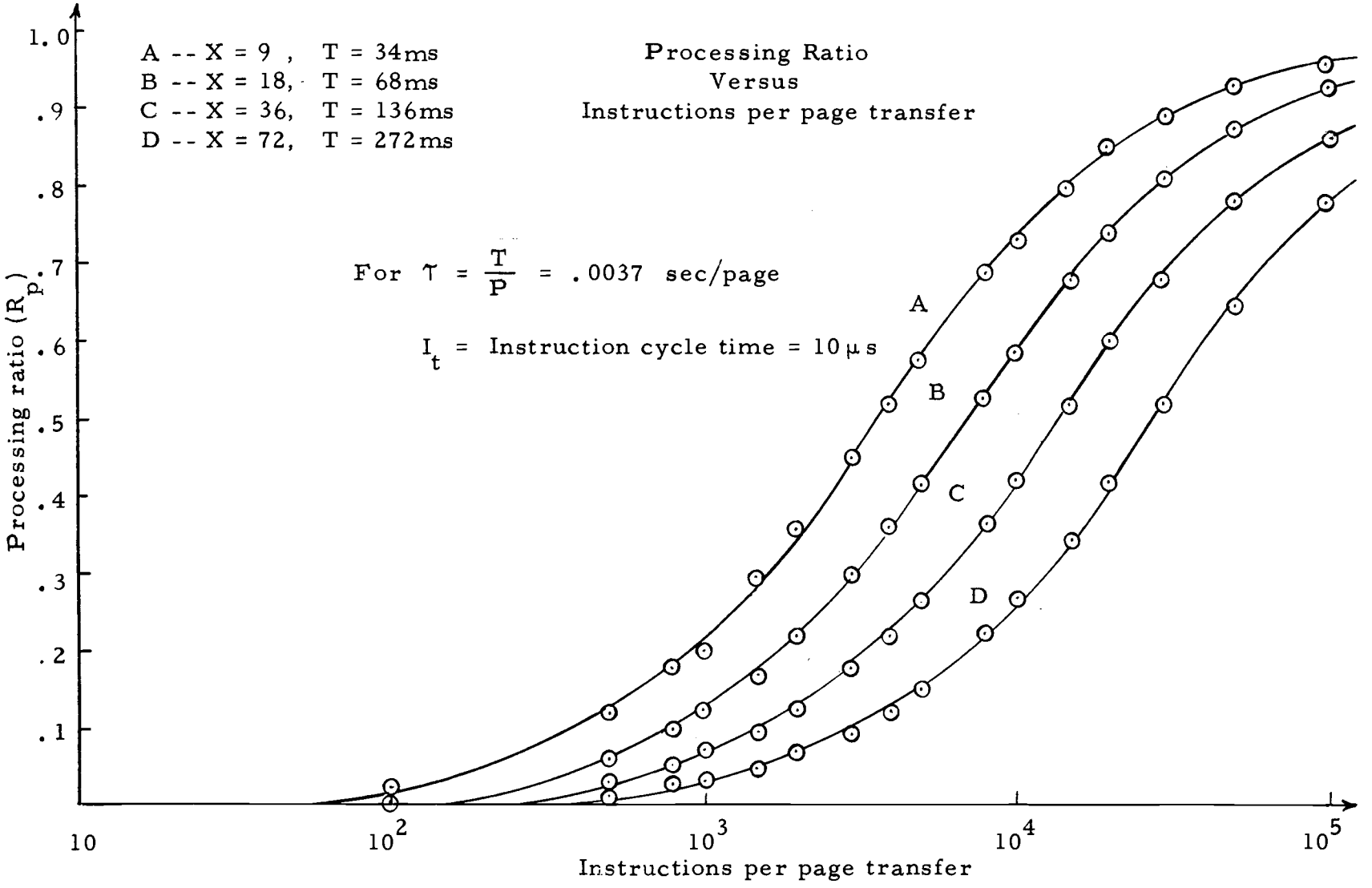



Figure 9. Processor ratio as a function of instructions per page transfer for the four cases of Figure 8.

processor ratio approaches its upper limit of unity.

## Effect on Processor Ratio of Varying the Page Size While Holding the Drum Storage Capacity Constant

The product of the page size (P) and the number of pages per track (X) will give the number of words which a single track is capable of storing. If this product is held constant, the effect on processor ratio of varying the page size may be determined. Figure 10 is an illustration of the variation on a single track basis. Table 2 lists the processor ratios for different values of operations per page transfer for each of the four cases shown in Figure 10. An increase in the number of pages per track by a factor of two corresponds to a decrease in page size by a factor of two.

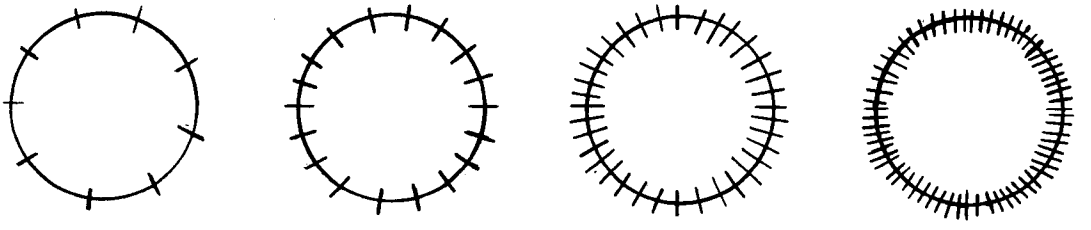



Figure 10. Four drum systems with the same capacity in words but different page sizes.

TABLE 2. Processor ratio as a function of operations per page transfer for the four cases of Figure 10.

| $E_i$ | X = 9* | X = 18* | X = 36* | X = 72* |
|---|---|---|---|---|
| | $R_p$ | $R_p$ | $R_p$ | $R_p$ |
| 1 | .00028 | .00029 | .00029 | .00029 |
| 10 | .00278 | .00285 | .00289 | .00291 |
| 100 | .02711 | .02782 | .02819 | .02838 |
| 500 | .12228 | .12517 | .12667 | .12743 |
| 800 | .18228 | .18629 | .18836 | .18941 |
| 1,000 | .21792 | .22250 | .22486 | .22606 |
| 1,500 | .29476 | .30033 | .30320 | .30465 |
| 2,000 | .35785 | .36400 | .36716 | .36876 |
| 3,000 | .45531 | .46193 | .46532 | .46703 |
| 4,000 | .52709 | .53373 | .53711 | .53882 |
| 5,000 | .58215 | .58867 | .59191 | .59357 |
| 8,000 | .69032 | .69599 | .69886 | .70030 |
| 10,000 | .73590 | .74105 | .74365 | .74496 |
| 15,000 | .80693 | .81105 | .81313 | .81417 |
| 20,000 | .84786 | .85127 | .85298 | .85384 |
| 30,000 | .89315 | .89367 | .89694 | .89757 |
| 50,000 | .93303 | .93468 | .93550 | .93592 |
| 100,000 | .96535 | .96624 | .96668 | .96690 |

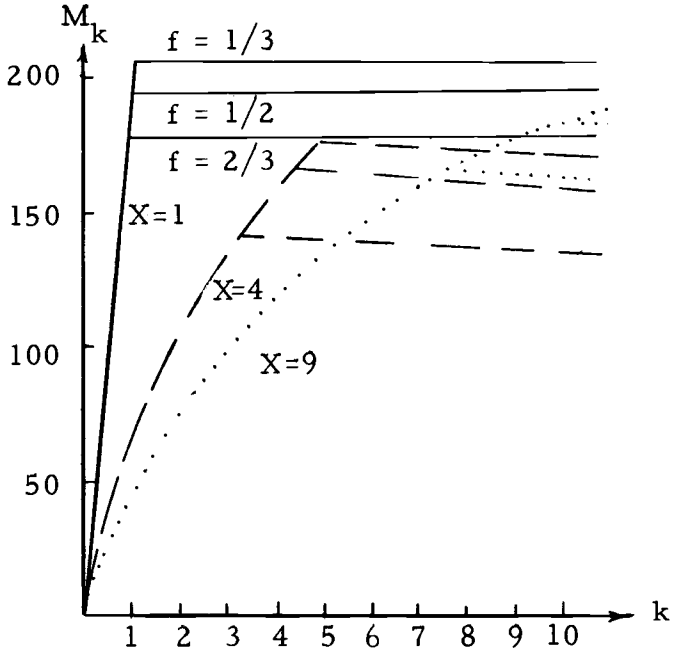
* This table is calculated for an instruction cycle time ($I_t$) of 10μs. and a drum revolution time (T) of 34ms.

As seen in Table 2, there is a slight increase in the processor ratio as the page size is decreased. However, it should be pointed out that the increase in the number of pages which accompanies this decrease in page size will require more housekeeping and updating on the part of the executive. Therefore the slight gain in processor ratio is partially cancelled by added system requirements.

Although it has been shown that the page size has little effect on the processor ratio, it should be mentioned that this parameter has a pertinent effect on the average number of pages which the drum can deliver in one revolution. The importance of this deliverable page rate is that a minimum allowable mean free path can be calculated by dividing the number of deliverable pages into the drum revolution time. In support of this work, Figures 11 and 12 show the results obtained by Lauer (1967), in which the deliverable page rate ($M_k$) and the minimum allowable mean free path ($T/M_k$) are given as a function of the number of page requests (k). The development of the equations from which these plots are based is given in Appendix B.

## The Effect of Transfer Time ($\tau$) on the Processor Ratio

The transfer time ($\tau = T/X$) may be decreased by either increasing the number of pages per track or decreasing the drum cycle time. The former method was examined in the prior section

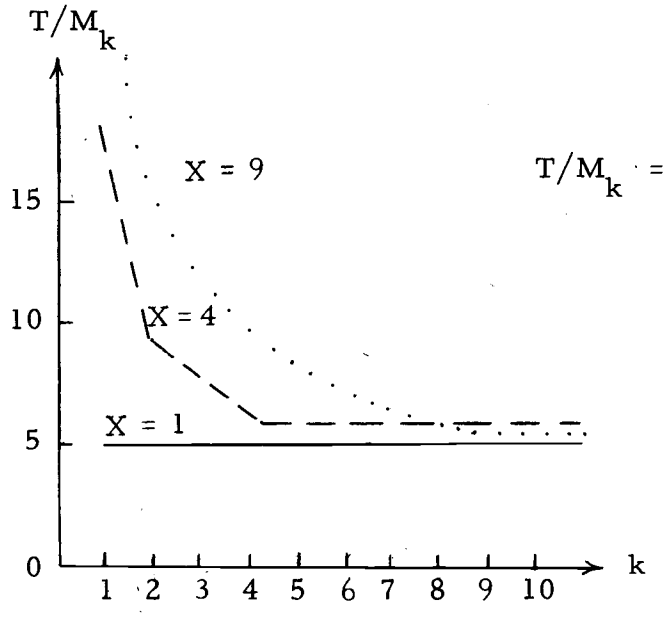



$M_k$ = Deliverable page rate

k = Number of requests in drum queue

f = Probability that a page must be written out to make room

Figure 11. Deliverable paging rate as a function of the number of requests in drum queue.




$T/M_k$ = Minimum mean free path (μ sec.)

Figure 12. Minimum allowable mean free path.

and did not have a very influential effect on the processor ratio. The latter method should yield a more significant effect, since the waiting time can be reduced by a faster drum system.

Figure 13 shows a family of curves of processor ratio versus instructions per page transfer for various transfer times. The processor ratio increases as the transfer time decreases. The relative margin of increase is dependent on the number of instructions per page transfer.

Although decreasing the drum cycle time is a better method of increasing the processor ratio than the method of decreasing the page size, there is also a cost factor which is usually higher for a faster system. Therefore a compromise should be made between the increase in processor ratio and the additional cost required to speed up the drum system.

## The Effect of Instruction Cycle Time on Processor Ratio

The instruction cycle time ($I_t$) is the average time necessary to perform a memory reference and complete the operation specified by the instruction operation code. Since the variable $E_i$ pertains to the number of instructions per page transfer, there is no need to include page transfer time within instruction cycle time. In Figure 14, processor ratio is shown in a function of instruction cycle time for different values of $E_i$.

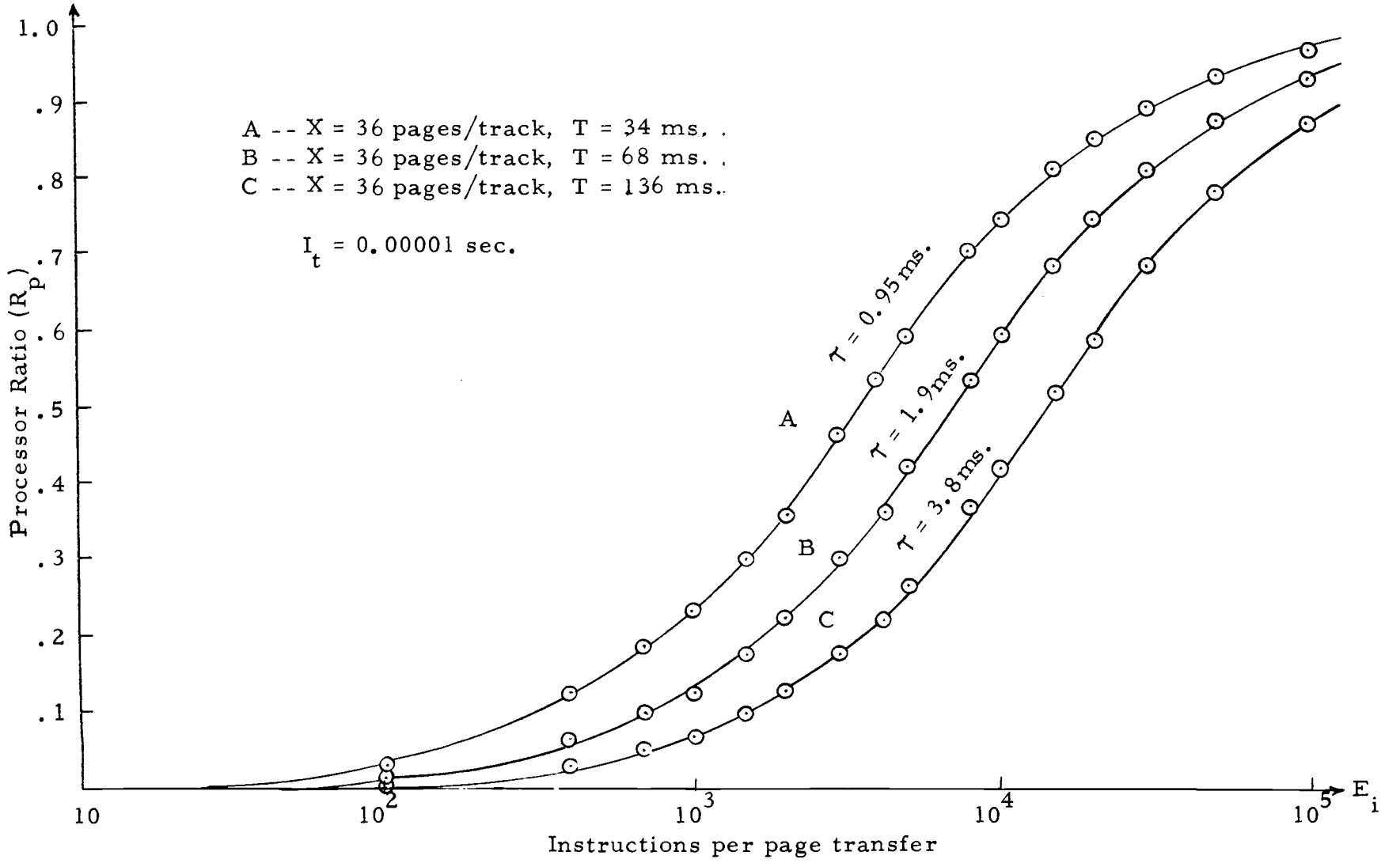



Figure 13. Processor ratio ($R_p$) as a function of instructions per page transfer ($E_i$) for different values of transfer time($\tau$).

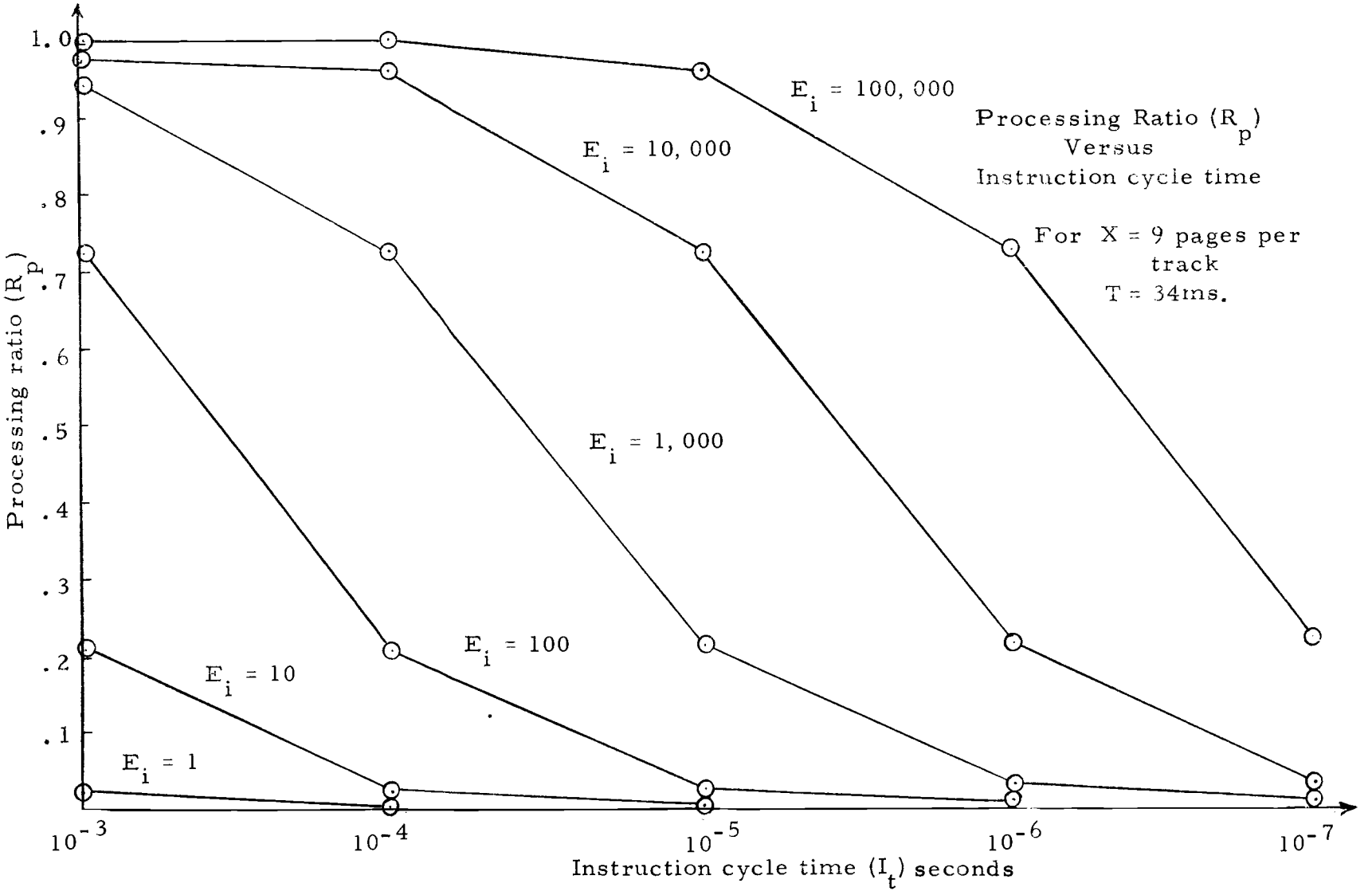



Figure 14. Processor ratio as a function of instruction cycle time.

The result is a decrease in processor ratio as the instruction cycle time is decreased. Therefore a slow computer system will have a better processor ratio than a faster system. This fact can be supported by the theory that a slower system will yield more time for page transfers to take place, since it takes longer to perform a given number of instructions. Therefore more processing time is available to overlap the time needed to swap pages between auxiliary memory and core. The overall effect is a decrease in the CPU idle time causing an increase in the processor ratio. The objective, however, is to process at a fast speed with a tolerable processor ratio.

In Figure 14 the curve corresponding to $E_i$ = 1000 instructions per page transfer at $I_t$ = 100 μsec. will yield a processor ratio of 0.73. If the speed of the system is increased by a factor of 10 making $I_t$ = 10 μsec., the processor ratio is reduced 70% to a value of 0.21.

From Figure 14, one can state that there is a tradeoff between speed and processor ratio in a paging system. To determine the best operating conditions of the sytem, one must examine the number of instructions per page transfer $E_i$. For systems which have programs characterized by large values of $E_i$, the speed of the system can be increased and the effect on processor ratio is less significant. For instance, in the case where $E_i$ = 100, 000 operations per page transfer, decreasing $I_t$ from 1 msec. to 1 μsec. would correspond to a reduction in processor ratio from 0.99 to 0.73. This means

that increasing the speed of the system 1000 fold would decrease the efficiency 27%. Such a tradeoff would be practical.

On the other hand, for $E_i = 1000$ operations per page transfer, the same increase in speed would result in a decrease of 97% in processor ratio. This tradeoff would be quite impractical.

For systems characterized by small values of $E_i$, generally the processor ratio is small. The fast systems have extremely small processor ratios and a decrease in speed to raise the processor ratio to a tolerable level would be advantageous in this case.

Therefore, in general, the tradeoff between speed and processor ratio is determined by the number of operations per page transfer.

## The Interdependence of Hardware and Software Parameters

The processor ratio has been used as a means for measuring the efficiency of a paging system. This ratio has been examined for variations in drum size, page size, page transfer time, and instruction cycle time. Each of these hardware parameters has been shown to be influential in some manner on the performance of the paging system. Yet they do not encompass a complete parameter space because their effect on the efficiency or processor ratio is dependent on a software parameter. This software parameter is the average number of instructions per page transfer. To estimate this parameter some knowledge of the characteristic addressing patterns of

the program repertoire is necessary. Once the parameter is estimated, the design of the system can be initiated.

To clearify the use of the interdependence of hardware and software parameters, let three distinct software environments be characterized as follows:

1. High school and college students running programs on remote teletype consoles.
2. Data processing or business programs bounded by input-output requirements.
3. Research or industrial programs from experienced users.

Since the users in case 1 are predominately novices, the addressing patterns of this group would probably be random and sequential with the former dominating. Case 2 would likely be characterized by all three patterns, but the sequential would tend to outweigh the others. In case 3 the researchers might use procedures and subroutines more frequently, which would lean to recursive patterns.

In each of these cases if the expected number of instructions per page transfer is estimated, the effect of the hardware parameters on the processor ratio may be determined. For instance, suppose that a cost analysis determined that the system could afford a core ratio of 0.25. For the random environment of case 1, the expected number of instructions per page transfer would be slightly greater than one instruction. For this case no combination of hardware

parameters would yield a tolerable processing ratio. Therefore the designer would either have to increase the core ratio or dismiss the thought of paging. From Figure 4 it can be seen that the core ratio should be increased to at least 0.7 to average 8 instructions per page transfer, but this core ratio is impractical and the paging process is dismissed.

It should be emphasized that the previous conclusion was reached on the assumption that the addressing patterns were completely random and the core allocated to the program was less than the storage required for its operation. In this situation a memory composed totally of random-access core storage would give the best performance. However, it should be mentioned that many type A users will have short programs. Therefore if a paging environment is necessary, a practical system would dictate that this type of user be limited to one or two pages.

For the sequential environment of case 2, Figure 5 shows that with a core ratio of 0.25 the number of instructions per page transfer can vary from 5 to 1,365 for page sizes from 4 to 1,024 words respectively. Table 2 indicates that the effect of increasing the page size has little influence on the processor ratio with respect to the drum configuration. Therefore the larger page size appears appropriate. To select the number of pages per track, Figure 9 indicates that the smaller values give a higher processing ratio. Therefore

the hypothetical drum of only one page per track would work best. The instruction cycle time would be a flexible parameter since high values of $E_i$ yield practical tradeoffs between speed and processor ratio as illustrated in Figure 14. The last parameter to affect the processor ratio is the drum cycle time, and as shown in Figure 13 the cycle time should be reduced to give a lower page transfer rate.

At this point the system has the following requirements for optimal performance assuming the addressing patterns are completely sequential:

1. Page size (P) increased to give larger number of instructions per page transfer ($E_i$).
2. Number of pages per track (X) decreased to give higher processing ratio.
3. The instruction cycle time ($I_t$) increased to give higher processing ratio.
4. The drum cycle time (T) decreased to give a higher processing ratio.

Although these variations seem to improve the performance of the system, there is still the question as to whether or not the drum can perform adequately when each of these parameters approach their optimum values. If the drum has X pages per track and the cycle time is T seconds, the minimum allowable mean free path which the drum can service is T/X seconds per page request. However,

the mean free path of the system is given by $MFP = E_i \times I_t$ . Therefore the variation of the four parameters are constrained by the equation

$$E_i \times I_t \geq T/X .$$

The former procedure can be applied to recursive patterns with the additional requirement that the recursive neighborhood be smaller than the core. The objective of the procedure was not to arrive at a specific design, but rather to examine the effects and interaction of the hardware and software parameters on the performance of the system. The random and sequential cases are hypothetical limits, and it is pertinent that these cases be investigated in order to understand the variations between the limits.

# V. CONCLUSIONS

The concept of paging and virtual memory has been examined in this paper. With respect to systems characterized by large storage requirements, the process of paging has been shown to be superior to both program segmentation and the utilization of auxiliary memory under program control.

The software parameters associated with paging have been defined by means of the various addressing patterns. These parameters are randomness, sequentialness, and recursiveness. The sequential and recursive patterns are more applicable to paging than the random addressing patterns. However, since all programs possess to a certain degree each type, all three characteristic patterns were analyzed.

The expected number of instructions per page transfer of the random pattern is dependent on the core ratio. In the sequential case, $E_i$ is dependent on both the core ratio and the page size. In the recursive case $E_i$ is dependent on the length of the recursive pattern and the size of the recursive neighborhood as compared with core memory.

The processor ratio was used as the principle reference in determining the effectiveness of the paging system. Utilizing the drum unit as a typical media of auxiliary storage, the processor

ratio exhibited the following variations:

1. Processor ratio increased as the number of operations per page transfer increased.
2. For a fixed transfer rate and page size, the processor ratio decreased as the physical size of the drum increased.
3. For a fixed drum capacity in words, the processor ratio increased slightly as the page size decreased.
4. The processor ratio increased as the drum cycle time decreased.
5. The processor ratio increased as the instruction cycle time increased.

It should be emphasized that the principle measure of contemporary paging systems is through-put. The idea is to service and complete as many jobs as possible in the shortest amount of time at a reasonable cost. By examining the processor ratio one can see that through-put is not always increased by increasing the CPU speed. Although it may appear that the system's through-put has improved, in actually the cost associated with a faster CPU may be impractical due to the decrease in processor ratio.

In designing paging systems, hardware parameters cannot be determined independently of software parameters. As was shown, it is the product of the instruction cycle time $(I_t)$ and the expected number of instructions per page transfer $(E_i)$, a hardware and

software parameter respectively, which effects the performance of the system. This product is termed the mean free path and both the program repertoire and the system characteristics determine its duration. If present-day paging systems are to operate efficiently, some means of measuring this hybrid parameter must be established between the limits of randomness, sequentialness, and recursiveness.

# BIBLIOGRAPHY

Abate, Joseph, Harvey Dubner and Sheldon B. Weinberg. 1968. Queuing analysis of the IBM 2314 disk storage facility. Journal of the Association for Computing Machinery 15:577-589.

Adiri, I. and B. Avi-Itzhak. 1969. A time-sharing queue with a finite number of customers. Journal of the Association for Computing Machinery 16:315-323.

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

Belady, L. A. and C. J. Kuehner. 1969. Dynamic space-sharing in computer systems. Communications of the Association for Computing Machinery 12:282-288.

Belady, L. A., R. A. Nelson and G. S. Shedler. 1969. An anomaly in space-time characteristics of certain programs running in a paging machine. Communications of the Association for Computing Machinery 12:349-353.

Coffman, E. G. 1969. Analysis of a drum input/output queue under scheduled operation in a paged computer system. Journal of the Association for Computing Machinery 16:73-90.

Corbato, F. J. and V. A. Vyssotsky. 1965. Introduction and overview of the Multics system. In: Fall Joint Computer Conference of the American Federation of Information Processing Societies, 1965. Vol. 27. Part I. Washington, D. C. p. 185-196.

Feller, William. 1957. An introduction to probability theory and its applications. 2d ed. Vol. 1. New York, John Wiley and Sons. 461 p.

Fikes, Richard E., Hugh C. Lauer and Albin L. Vareha. 1968. Steps toward a general-purpose time-sharing system using large capacity core storage and TSS/360. In: Proceedings of the 23rd National Conference of the Association for Computing Machinery, 1968. Princeton, N. J. p. 7-18. (A.C.M. Publication P-68) 1968.

Fine, Gerald H., Calvin W. Jackson and Paul V. McIsaac. 1966. Dynamic program behavior under paging. In: Proceedings of the 21st National Conference of the Association for Computing Machinery, 1966. Washington, D. C. p. 223-228. (A. C. M. Publication P-66)

Flores, Ivan. 1967a. Virtual memory and paging. Part I. A review with examples. Datamation 13: 31-34. August.

_____ 1967b. Virtual memory and paging. Part II. Dynamic relocation. Datamation 13:41-48. September.

Gibson, D. H. 1967. Considerations in block-oriented system design. In: Spring Joint Computer Conference of the American Federation of Information Processing Societies, Atlantic City, 1967. Vol. 30. Washington, D. C. p. 75-80.

Howarth, D. J., R. B. Payne and F. H. Sumner. 1961. The Manchester university Atlas operating system. Part II. User's description. The Computer Journal 4: 226-229.

Kilburn, T. et al. 1961. The Manchester university Atlas operating system. Part I. Internal organization. The Computer Journal 4: 222-225.

Kilburn, T. et al. 1962. One-level storage system. The Transactions on Electrical Computers of the Institute of Radio Engineers 11: 223-235.

Laski, J. G. 1968. Segmentation and virtual address topology - an essay in virtual research. The Computer Journal 11: 35-40.

Lauer, Hugh C. 1967. Bulk core in a 360/67 time-sharing system. In: Fall Joint Computer Conference of the American Federation of Information Processing Societies, Anaheim, 1967. Vol. 31. Washington, D. C. p. 601-609.

Lowe, Thomas C. 1969. Analysis of boolean program models for time-sharing, paged environments. Communications of the Association for Computing Machinery 12: 199-205.

McKellar, A. C. and E. G. Coffman. 1969. Organizing matrices and matrix operations for paged memory systems. Communications of the Association for Computing Machinery 12: 153-165.

Sheperd, M. J. and A. J. Willmott. 1968. Cluster analysis on the Atlas computer. The Computer Journal 11:57-62.

Stimler, Saul. 1969. Some criteria for time-sharing system performance. Communications of the Association for Computing Machinery 12:47-53.

Wallace, V. L. and D. L. Mason. 1969. Degree of multiprogramming in page-on-demand systems. Communications of the Association for Computing Machinery 12: 305-308, 318.

# APPENDICES

# APPENDIX A

## THE MEAN WAITING TIME

Let the amount of time which a task spends waiting for a single page be a function of the rotational delay of the drum. Then the waiting time is given by the equation

$$\overline{W} = t_1 + (j \times \tau) \ .$$

Where $t_1$ is a uniformly distributed random variable representing the time to the start of the next channel program,

$$0 \leq t_1 \leq T.$$

$j$ is the slot position of the page requested, uniformly distributed over the interval $(1, X)$,

$$1 \leq j \leq X \ .$$

$\tau$ is the transfer time of a page,

$$\tau = T/X \ .$$

Then the mean waiting time is given by the equations

$$\overline{W} = \overline{t_1} + (\overline{j} \times \overline{\tau})$$

$$\overline{W} = T/2 + (\frac{X+1}{2})(T/X)$$

$$\overline{W} = T(\frac{2X + 1}{2X}) = T(1 + \frac{1}{2X}) \ .$$

# APPENDIX B

## DERIVATION OF THE DELIVERABLE PAGE RATE AND THE MINIMUM MEAN FREE PATH

The expected rate at which pages are read from the drum is a function of k, the number of tasks in the drum queue, and f, the probability that a page must be written on the drum to make room in core. It is possible that $f \leq 1$, since some pages in core have valid copies on the drum already.

Then for each page requested the drum must handle an average of 1+f pages. Thus the maximum number of requests which the X-slot drum can handle is $\frac{X}{1+f}$ pages per revolution or $\frac{X}{(1+f)T}$ pages per second. Because of slot conflicts, this rate can only be approached. A classical probability model known as the Urn-model Occupancy problem (Feller, 1957) wasused by Lauer to give the result. With X slots and k requests, the probability $p_i$, that exactly i pages $[1 \leq i \leq \min(X, k)]$ can be read in one revolution is given by

$$P_i = \frac{X!}{(X-i)!\, i!} \sum_{v=0}^{i} (-1)^v \left(\frac{i!}{(i-v)!\, v!}\right)\left(\frac{i-v}{X}\right)^k .$$

The average or expected number of pages which the drum can read,

given $k$, is

$$M_k = \sum_{i=1}^{k} ip_i .$$

Therefore the minimum allowable mean free path is $T/M_k$.

# APPENDIX C

## GLOSSARY

Instruction cycle time. The average time necessary to perform a memory reference and complete the operation specified by the operation code.

Mean free path. The average time between consecutive page transfers. The product of the number of instructions per page transfer and the instruction cycle time.

Memory address register. Register which points to block and line numbers currently utilized in core.

Page address register. Register which contains labels of pages and their associated core block locations.

Processing ratio or processor ratio. Ratio of processing time to total time for a particular task. $R_p = T_p/T_t$.

Random addressing pattern. A sequence of CPU references to memory where at any point in the addressing pattern any address is equally likely to be referenced.

Recursive addressing pattern. A sequence of CPU references to memory where the references in the sequence are repeated either in a cyclic or haphazard manner.

Segment address register. Register which contains labels of pages and their associated auxiliary segment locations.

Sequential addressing pattern. A sequence of CPU references to memory where at any point in the addressing sequence a particular address is exactly one word away from the preceding address.

Transfer time. Time required to read a single page from the drum.