DOCUMENT RESUME

ED 101 694 IR 001 544

AUTHOR	Stellhorn, William Howard
TITLE	A Specialized Computer for Information Retrieval. Report No. 74-637.
INSTITUTION	Illinois Univ., Urbana. Dept. of Computer Science.
SPONS AGENCY	National Science Foundation, Washington, D.C.
REPORT NO	UIUCDCS-R-74-637
PUB DATE	Oct 74
NOTE	117p.
EDRS PRICE	MF-$0.76 HC-$5.70 PLUS POSTAGE
DESCRIPTORS	*Algorithms; *Computers; *Computer Storage Devices; *Data Bases; Data Processing; Documentation; Information Processing; *Information Retrieval; *Information Storage; Information Systems; Search Strategies; Technological Advancement

ABSTRACT

Response time in large, inverted file document retrieval systems is determined by time required to access files of document identifiers on disk and process a Boolean search request. A specialized computer system has been devised that can perform a complicated sample search involving 70 terms and over 60,000 document references 12 to 60 times faster than a conventional machine. Many small searches can be processed concurrently with little effect on system performance. The system can be realized with currently available technology and has been tried in numerous simulations involving various system configurations and other factors. (SK)

Report No. UIUCDCS-R-74-637

A SPECIALIZED COMPUTER FOR INFORMATION RETRIEVAL*

by

William Howard Stellhorn

October 1974

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation under Grant No. US NSF-GJ-36936 and was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science, October 1974.

A SPECIALIZED COMPUTER FOR INFORMATION RETRIEVAL

William Howard Stellhorn, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1974

Response time in large, inverted file document retrieval systems is determined primarily by the time required to access files of document identifiers on disk and perform the processing associated with a Boolean search request. This paper describes a specialized computer system capable of performing these functions in hardware. Using this equipment, a complicated sample search involving 70 terms and over 60,000 document references can be performed from 12 to 60 times faster than with a conventional machine, and many small searches can be processed concurrently with very little effect upon system performance.

A detailed description of the system, which can be realized with currently-available technology, is presented; and algorithms for controlling the progress of a search are discussed. Results from numerous simulations involving various system configurations and other factors are also reported.

ACKNOWLEDGMENT

Several persons have contributed significantly to the success of this project, and their help is gratefully acknowledged. Special thanks are due to my advisor, Professor David J. Kuck, for his constant enthusiasm and continuing help and support.

Thanks, too, to officials at the National Library of Medicine, especially Messrs. W. H. Caldwell and D. B. McCarn, for information concerning the MEDLARS and MEDLINE systems.

Much of the test data reported here was obtained through the efforts of Mr. B. J. Hurley in maintaining, modifying and running the simulation program.

Finally, the financial support of the Department of Computer Science and the typing and drafting services of Mrs. Vivian Alsip and Messrs. S. Zundo and R. Bright have been invaluable.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Overview

During the last few years, the growth of on-line information retrieval services has been rapid, and this expansion is expected to continue on a major scale for a long time to come. As such a system grows and prospers, two problems often arise. First, the data base tends to grow--rapidly, sometimes--and it is often difficult both to justify deleting old material and to select items to be discarded. Second, the number of users desiring service may also tend to increase. Both of these developments increase the load on the system, until eventually it becomes difficult to provide sufficiently fast response to satisfy on-line users.

A number of systems already in operation are large enough to experience these problems, and many have prospects for nearly unlimited growth. To cite a single example, it is reported [1, 2] that Mead Data Central, Incorporated's LEXIS (formerly OBAR) now contains the full text of all New York and Ohio statutes and supreme and appellate court decisions plus all United States Supreme Court decisions and a number of other federal materials. The complete United States Code and all federal court of appeals and district court decisions are to be available in the spring of 1974. This data base contains well over 100 million words of English text and grows at a rate of several million words per year, and the nature of the material makes deletion of old documents unacceptable. Besides expanding the coverage of its data base, Mead is said to be planning to offer retrieval services in a number of states not already served.

Other large retrieval systems include those maintained by the

National Library of Medicine [3], to be discussed in more detail later, and by the United States Patent Office [4].

This report describes a specialized hardware subsystem for performing the time-consuming term access and coordination functions in large, inverted file, document retrieval systems. It is conservatively estimated that the time to perform these functions for a large search involving 70 terms can be reduced by factors between 12 and 60 depending upon the size of the hardware system employed. The speed-up is not so great for a smaller search involving only a few terms; but in this case, a number of searches can be performed in parallel with very little effect upon the system, so that the average elapsed time per search can still be reduced dramatically.

The remaining section of Chapter 1 describes the organization of inverted file retrieval systems and identifies that portion of their operation which the proposed hardware will perform. Chapter 2 describes the hardware components in detail, analyzes the timing constraints imposed by each and shows that several processors of different sizes and capacities can be built using currently-available subsystems and logic devices. Chapter 3 describes several fundamental software procedures which must be provided to control the operation of the hardware and presents details of the processing algorithms which have been used in simulating the proposed system. Chapter 4 presents results of a large number of simulation experiments in which the performance of the system has been evaluated in a realistic retrieval situation. These tests are based on parameters of actual searches which could be performed in a particular, large, operational document retrieval system. Variations in the capacity of the hardware, the size of the data memory, the size of the data base, and several other

factors are considered. A few results which illustrate the potential of the system to process multiple independent searches simultaneously are also discussed. Conclusions are presented in Chapter 5.

1.2 Operational Environment

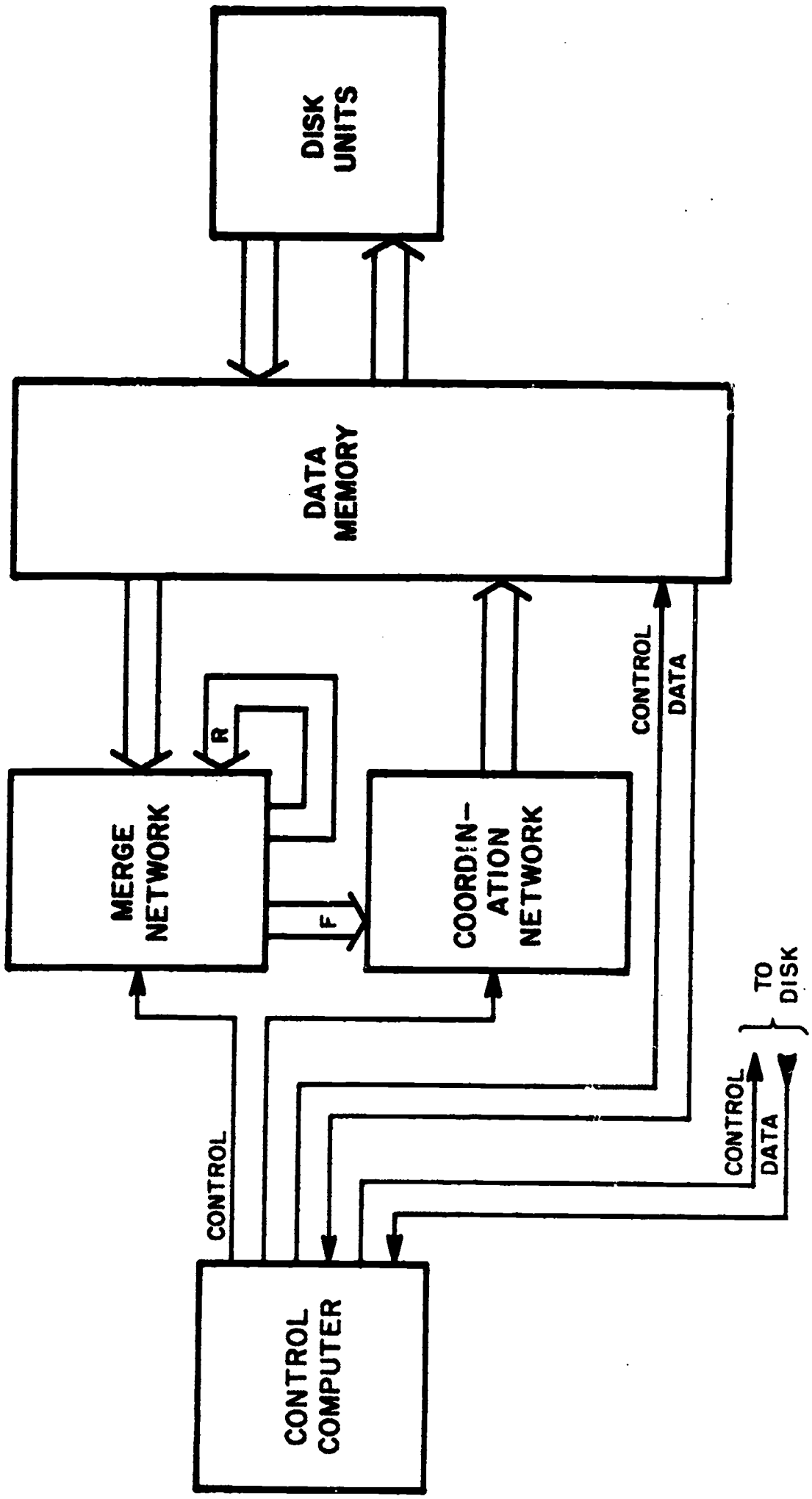
Nearly all the mechanized document retrieval systems currently in operation employ inverted files for data base organization. Certain index terms (possibly all the information-bearing words in the original text) are selected as descriptors for each document in the system. Each index term is entered into a directory, the index file, along with certain information including a pointer into a second directory, the postings file, which contains a list of all the contexts (documents or document subdivisions) identified by the index term in question. To request information from such a system, a user provides a list of index terms and specifies the Boolean relationships (OR, AND, AND NOT) among them which must be satisfied in any document that is retrieved. The system then consults the index file to obtain the required postings file addresses, reads the postings lists, and coordinates them, i.e., selects from them those context identifiers which satisfy the search logic. This last procedure requires at least one disk access per search term and, if there is a large number of search terms or if some of the associated postings lists are very long, it may require a substantial amount of central processor time as well. The new system described in this report accepts a list of postings file addresses and performs the access and coordination operations automatically, at disk speeds.

2. TERM COORDINATION HARDWARE

This chapter describes in detail the proposed hardware for inverted file processing. Section 2.1 contains a brief general description of the system and the functions of its various components. Section 2.2 presents a fairly detailed example of its operation, illustrating the parallel nature of the design and some of the timing constraints which must be satisfied. Hardware requirements are presented in detail, along with logic designs for the critical components, in section 2.3. Section 2.4 contains a systems-oriented analysis, showing how the components interact with one another and how their activity is distributed within the available time. Several design alternatives are discussed, and the associated limitations are identified. Section 2.5 summarizes the principal results of the chapter.

2.1 System Description

Term coordination in inverted file systems can be performed almost entirely by hardware operating at disk speeds using the configuration shown in Figure 2.1. Suppose the search "L1 OR L2" is to be performed, i.e., two ordered lists, L1 and L2, are to be merged into a single ordered list with duplicate elements removed. Suppose further that L1 will be available for reading before L2. L1 and L2 are initially stored on disk in n-word blocks. When L1 becomes available, it is read into data memory and held there until L2 comes under the read heads. Merging and coordination (selection of the desired elements from the merged list) proceed in parallel with the reading of L2, and the entire operation is completed shortly after the last block of L2 has been read. The output list may be retained in data memory if



F ure 2.1. Hardware Config uration

space is available or written back on disk while the coordination procedure continues.

The heart of the system is the merge network, an "odd-even" merge of the type proposed by Batcher [5-7]. The basic building block of this network is the comparison element, Figure 2.2, which accepts two input numbers and routes their minimum and maximum to its "MIN" and "MAX" output terminals, respectively. Batcher shows how these elements can be combined to form a hardware merge network whose input is two n-element ordered lists and whose output is a single ordered list containing 2n elements.

The coordination network selects from the output of the merge network those elements which satisfy the Boolean logic specified in the search request, and returns the edited list to the data memory. This function is accomplished by comparing adjacent terms on the merged list and accepting or rejecting terms as required.

To match the high speed parallel processing capabilities of the merge and coordination networks, it is necessary to provide a wide-band data memory and a disk system, preferably equipped with a hardware queuer, which has the capability of reading simultaneously from n tracks while at the same time writing simultaneously on n other tracks. Such a disk is currently in use with the Illiac IV computer.

The function of the control computer during the merging operation is one of supervision and bookkeeping. It provides memory management services and guarantees that data are routed to the merge network and to the disk in the proper sequence.

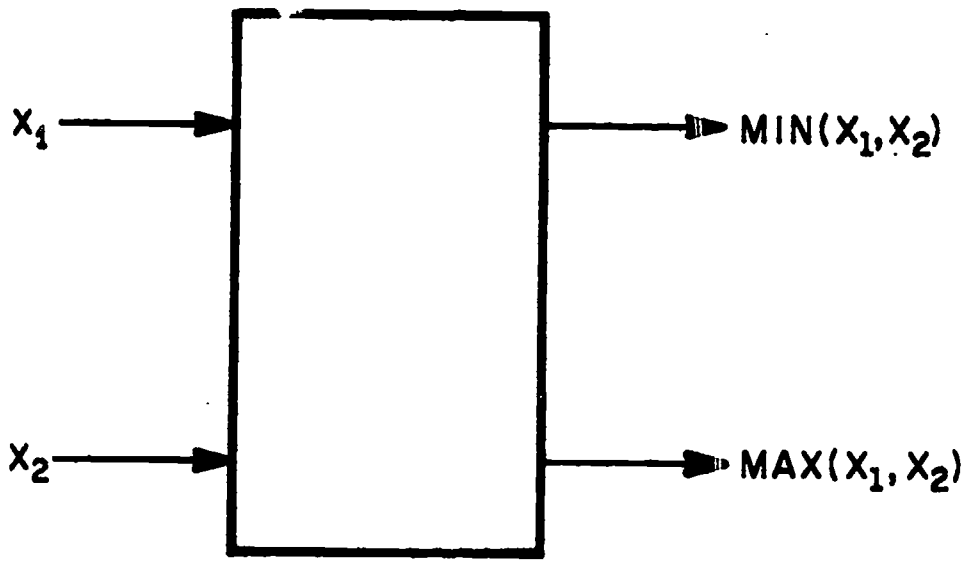


Figure 2.2. Symbol for a Comparison Element

2.2 Example

As an example of the operation of the system consider the two lists

L1: 3, 5, 8, 10, 12, 27

L2: 2, 3, 5, 6, 10, 13, 25, 27, 33

and the search request "L1 OR L2". Again assume that L1 is available from the disk before L2. Let n=3, so that three elements from either list may be read or written simultaneously.[1] This effectively divides the two original lists into five sublists

L11: 3, 5, 8

L12: 10, 12, 27

L21: 2, 3, 5

L22: 6, 10, 13

L23: 25, 27, 33.

Let one hardware cycle be defined as the time required to read or write one such n-word sublist, i.e., the time required for a conventional disk to transmit one computer word. This is the amount of time available for one complete processing sequence in the hardware. The term cycle will be used in this sense throughout this report except where a different meaning is specified explicitly or is clearly intended from context.

The first step in processing the sample search request is to read L1 into data memory. Then, when L2 becomes available, the actions described below and summarized in Table 2.1 occur during successive hardware cycles. In the table, Lij refers to sublist j of input list i, LRj refers to sublist

---

[1] Three is a convenient value of n to use for purposes of illustration. In practice, because of the requirements of the merge network, n must be a power of two.

CYCLE	PARALLEL PROCESSES: READ	MERGE	COORDINATE	RETURN TO MEMORY	WRITE (OPTIONAL)
1	L21(2,3,5)	R0(0,0,0) → ---------- L11(3,5,8) R1(3,5,8)			
2	L22(6,10,13)	R1(3,5,8) → F2(2,3,3) L21(2,3,5) R2(5,5,8)	F2(2,3,3)		
3	L23(25,27,33)	R2(5,5,8) → F3(5,5,6) L22(6,10,13) R3(8,10,13)	F3(5,5,6)	LR1(2,3,5)	
4		R3(8,10,13) → F4(8,10,10) L12(10,12,27) R4(12,13,27)	F4(8,10,10)	LR2(6,8,10)	LR1(2,3,5)
5		R4(12,13,27) → F5(12,13,25) L23(25,27,33) R5(27,27,33)	F5(12,13,25)	LR3(12,13,25)	LR2(6,8,10)
6		R5(27,27,33) → F6(27,27,33) ---(H,H,H) R6(H,H,H)	F6(27,27,33)	LR4(27,33,H)	LR3(12,13,25)
7					LR4(27,33,H)

Table 2.1. Processing Summary for Term Coordination Example

j of final result, and Fj and Rj refer to the cycle j merge outputs at F and R as shown in Figure 2.1. F is passed to the coordination system and R is returned to the merge network for further processing. The letter H denotes the computer word (111...111), which is used as a filler to provide all postings lists with an integral multiple of n entries.

Processing proceeds as follows:

Cycle 1. A. Read L21.

B. Set R=(0,0,0), and merge R with L11. This will produce outputs F1=(0,0,0) and R1=L11. Ignore F1.

Cycle 2. A. Read L22.

B. Merge R1 (L11) and L21. Result F2 contains the n (three) smallest elements in the combined list, and can be passed to the coordination network. Result R2 is returned to the merge network for further processing. Note that R2 does not contain the data element 6, which should be on the next sublist of the merged result.

C. Coordinate F2, i.e., eliminate the duplicate element 3.

Cycle 3. A. Read L23.

B. Compare the first element of L12 with that of L22. It will be shown later that the smaller of these determines which sublist should be transmitted next to the merge network. In this case, the answer is L22.

C. Merge R2 with L22.

D. Coordinate terms in F3 and combine with the previous result to form the completed sublist LR1(2,3,5) and the partial result: 6. Return LR1 to data memory.

Cycles 4 and 5 proceed as cycle 3 except that no new data are read from disk. If the output is to be returned to disk, writing can begin in cycle 4. During cycle 6, "high value" inputs are supplied internally to the merge network in order to force R5 to the F output terminals. The resulting F6 is then coordinated, padded with one "H" entry and returned to memory. If the result is being returned to disk, the last sublist is written during cycle 7.

In general, if lists L1 and L2 contain i and j sublists, respectively, then one new sublist is processed during each of the first (i+j) hardware cycles, one additional cycle is required to generate the last sublist of the result and return it to memory, and one extra cycle is needed if the result is to be written on disk. The total number of cycles required, t, is given by

$$t = i+j+w+1, \qquad (2.1)$$

where w is 1 if the result is written on disk and 0 otherwise.

2.3 Hardware Requirements

Wide variation is possible in the parameters of the proposed system, especially with respect to the degree of parallelism employed. A corresponding variation exists in the demands which are placed on the various system components and in the level of performance which can be achieved. This section defines hardware requirements in detail, proposes ways in which they can be satisfied, and identifies the factors which limit the de ‚n. The performance capabilities of various configurations are the subject of Chapter 4.

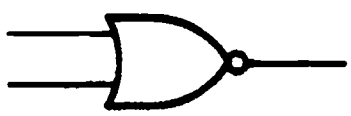
Throughout this analysis, the objective is to show that the required subsystems can be realized using currently-available technology and

that they can operate within the time constraints imposed by the process. Where detailed logic designs and their associated timings are discussed, standard ECL-10,000 components have been assumed [8], and many of the attractive characteristics of this family of devices have been employed. In particular, fast Exclusive-OR gates, the "wired OR" and the availability of both true and complement outputs from most devices have all been used. Propagation delays have been increased approximately by a factor of 2 (1.5 for shift register parallel input and output) from published typical values in order to produce a conservative design. Logic symbols and device characteristics employed in these designs are defined in Figure 2.3.

This discussion concentrates mainly on the characteristics of the largest system which is currently considered practical and useful. In some cases alternative designs are mentioned, especially where slower, cheaper components could be employed, but a detailed design optimization study is beyond the scope of this report.

The standard design chosen for study is a system with 256 parallel transmission paths throughout (n=256). Smaller systems would, of course, have correspondingly less stringent requirements: results in Chapter 4 indicate that a very powerful system can be built using only 16 parallel paths.

A head-per-track disk with the required parallel transmission facilities and with the other parameters shown in Table 2.2 has been assumed. The characteristics in the table are typical of a number of well-established disk units, and a head-per-track disk with parallel transmission facilities and approximately the required transfer rate has also been installed.

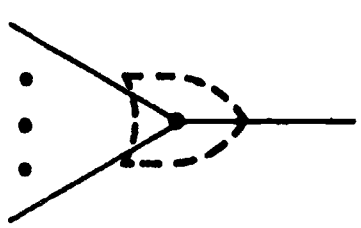


NOR Gate

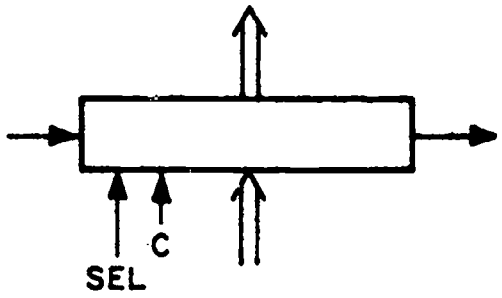
Propagation Delay: 5ns

Exclusive OR Gate

Propagation Delay: 7ns



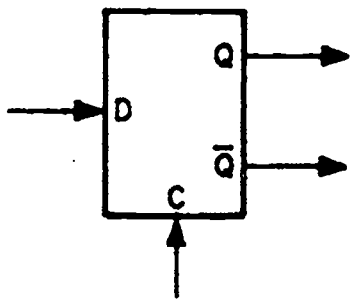
Implied (wired) OR



Shift Register

Propagation Delay: 6ns per operation

Operating Features: Parallel or serial input
Parallel or serial output
Left or right shift



Latch

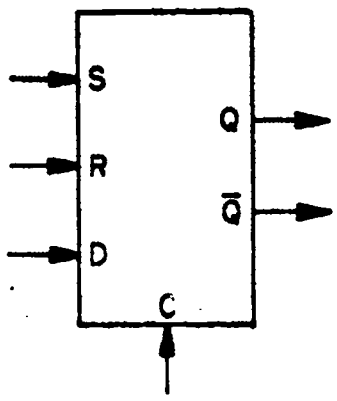
Propagation Delay: 7ns

Truth Table:

C	D	$Q_t+\Delta$
L	L	L
L	H	H
H	L	Q_t
H	H	Q_t

Outputs are latched on positive clock transitions

Figure 2.3. Logic Symbols and Device Characteristics



Type D, Master-Slave Flip-Flop

Propagation Delay: 7ns

Clocked Truth Table:

C	D	$Q_{t+\Delta}$
L	L	Q_t
L	H	Q_t
H	L	L
H	H	H

R-S Truth Table:

R	S	$Q_{t+\Delta}$
L	L	Q_t
L	H	H
H	L	L
H	H	N.D.

Clock "H" is positive clock transition

"N.D." means not defined

R and S inputs are independent of the clock

Figure 2.3 (continued). Logic Symbols and Device Characteristics

Disk rotation time	25. ms
*Word size	32. bits
Storage density	1800. words/physical track
**Tracks transmitted in parallel	256. physical tracks/logical track
Read time per sublist (one word per physical track)	13.89 μs
Transfer rate	2.30(10^6) bits/sec./physical track
	589.(10^6) bits/sec. for total 256-head parallel transmission

*Each document identifier in a postings file occupies one computer word.

**Values examined range from 1 through 512.

Table 2.2. Design Parameters for Standard System

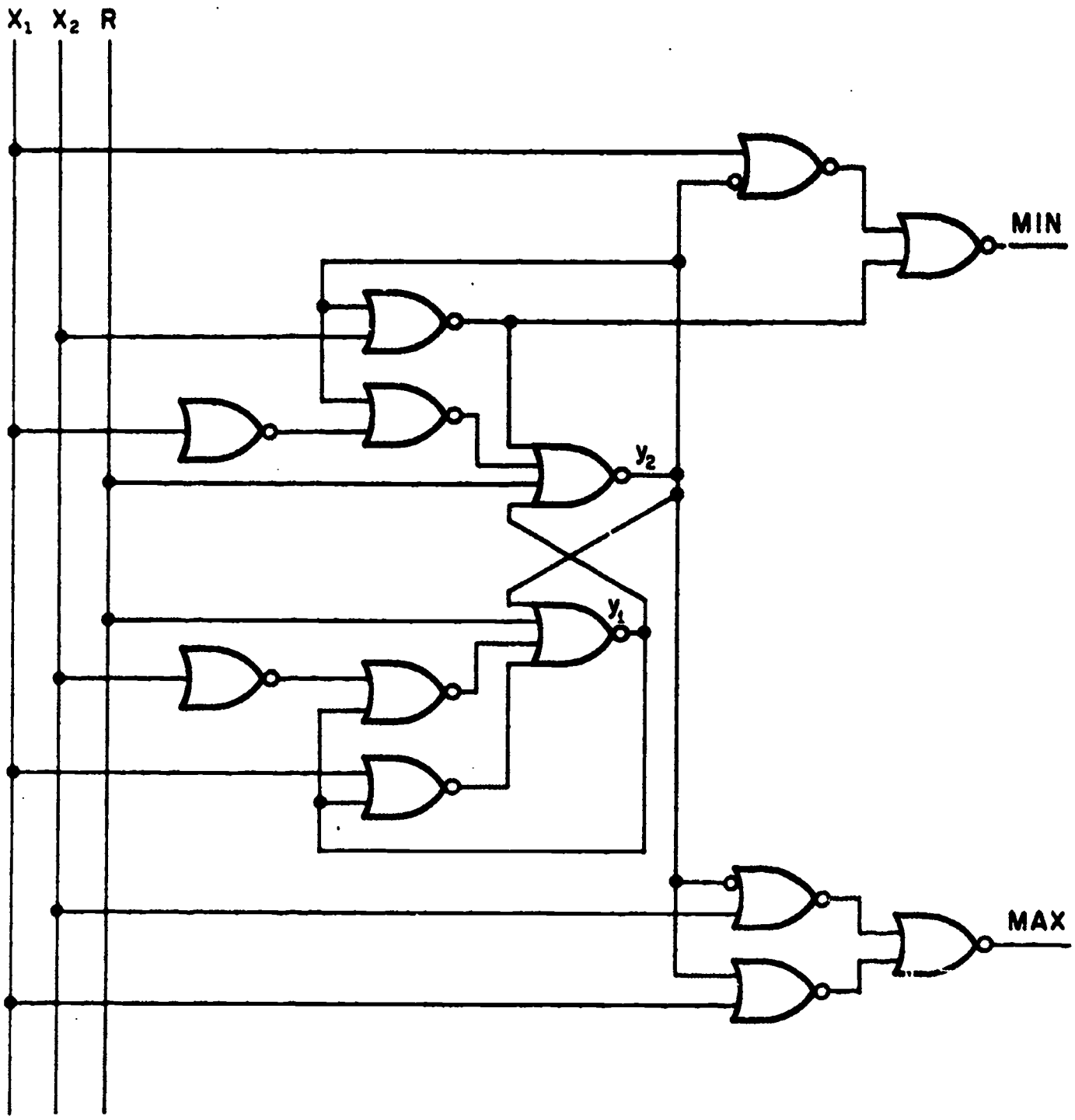
Postings files are assumed to be organized as n-word sublists stored in consecutive locations on one or more logical tracks. Each entry is one 32-bit word which uniquely identifies one document in the data base. No identifier may appear more than once on any given list except H, the "high value" filler, which may appear as many as n-1 times, but only on the last sublist in the file.

2.3.1 Merge Network

For this analysis, a merge network composed of bit-serial comparison elements is employed, and it is assumed that data items on each input list are arranged in nondescending order. In [5], Batcher gives a simple iterative rule for constructing odd-even merge networks of any desired size provided the number of elements, n, on each input list is a power of 2. He also shows that a $2^p \times 2^p$ merging network constructed according to this rule requires $p(2^p) + 1$ comparison elements and that the longest path through such a network contains $p + 1$ comparison elements.

Batcher states that a bit-serial comparison element can be implemented with 13 NORS, but does not give a specific design. One possible implementation is shown in Figure 2.4. An initial Reset signal, R, leaves $y_1 = y_2 = 0$. As long as $x_1 = x_2$, no change occurs, and the outputs are equal. As soon as x_1 and x_2 differ, y_1 and y_2 are changed to establish the appropriate output connections and locked into their new states until another reset signal is received.

The longest path through one comparison element contains seven gates and thus requires a propagation time of 35ns under the assumptions of



$$y_1 = \bar{R}\bar{y}_2 (x_1 + y_1)(\bar{x}_2 + y_1)$$
$$y_2 = \bar{R}\bar{y}_1 (\bar{x}_1 + y_2)(x_2 + y_2)$$
$$MIN = x_1 y_2 + x_2 \bar{y}_2$$
$$MAX = x_1 \bar{y}_2 + x_2 y_2$$

Figure 2.4. Logic Diagram for a Comparison Element

Figure 2.3. If 7ns latches are installed at the outputs of each comparison element, then new inputs may be accepted every 42 nanoseconds.

Table 2.3 lists the number of comparison elements, the gate counts and the maximum path lengths for network sizes of interest. The table also shows the time required to merge two, n-element lists of 33-bit words[2] in networks with and without latches at each stage.

The example in the previous section emphasized the fact that only half the output of the merge network (n terms) is available for coordination after each hardware cycle; the other half must be fed back into the network for comparison with the next input list. A group of n, 33-bit shift registers is required to collect the bit serial output from one cycle and present it for processing in the next, as illustrated in Figure 2.5.

Special inputs to the merge network are used during the first and last cycles of a merge procedure in order to force the first and last sublists to the proper output terminals. During the first cycle, the shift registers are cleared to 0 and then applied to the lower input terminals. Consequently, after the first cycle, the upper outputs are all zero and the lower outputs contain valid data from the upper inputs. During the last cycle of operation, the upper inputs are all set to 1 so that after that cycle is complete, the lower inputs appear at the upper outputs and all the shift registers at the lower outputs are filled with 1's.

---

[2]The thirty-third bit is supplied by the system as required for the "AND NOT" coordination algorithm to be described in section 2.3.2.

Input List Length	Comparison Elements on Longest Path	Total Comparison Elements	Total Gates	Merge Time for Two Lists of 33-bit Words Without Latches	With Latches
1	1	1	13	1.155 μs	1.386 μs
2	2	3	39	2.310	1.428
4	3	9	117	3.465	1.470
8	4	25	325	4.620	1.512
16	5	65	845	5.775	1.554
32	6	161	2,093	6.930	1.596
64	7	385	5,005	8.085	1.638
128	8	897	11,661	9.240	1.680
256	9	2049	26,637	10.395	1.722
512	10	4609	59,917	11.550	1.764

Table 2.3. Merge Network Characteristics

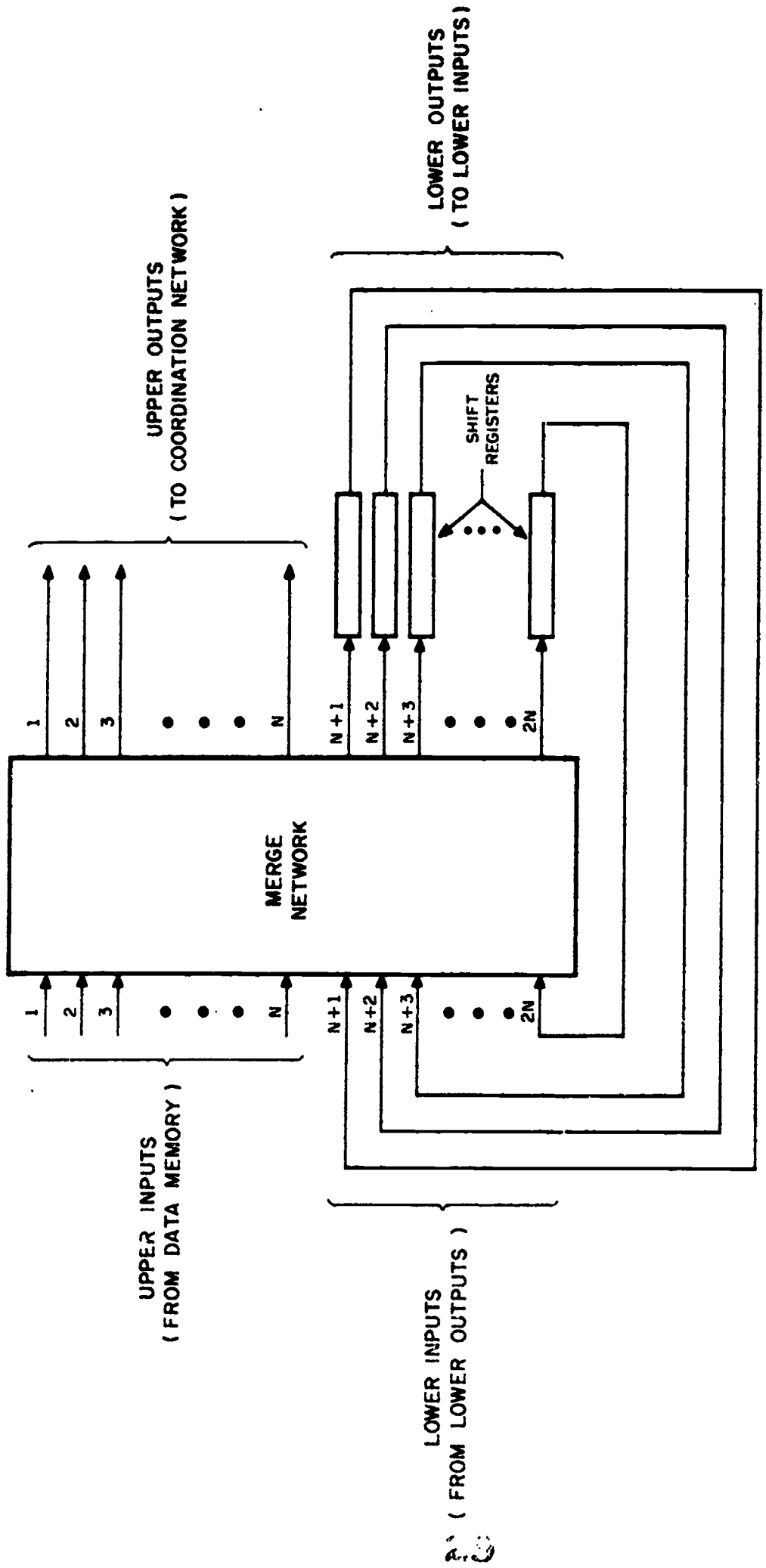


Figure 2.5. Merge Network Feedback Connections

2.3.2 Coordination Network

2.3.2.1 General Description

The function of the coordination network is to select from the output of the merge network those document identification numbers which satisfy the current search request.

Suppose that the current output of the merge network is

$$1_A,\ 1_B,\ 2_A,\ 3_A,\ 3_B,\ 4_B,\ 5_A,\ H_B,$$

where the subscripts indicate the list of origin and "H" represents the filler word which may occur at the end of a list. Then, the three allowable searches and the desired results are

A OR B = 1, 2, 3, 4, 5

A AND B = 1, 3

and

A AND NOT B = 2, 5.

In order to make this selection, the coordination network employs n identical logic circuits which compare adjacent postings as they arrive in bit serial form from the merge network, and generate the appropriate control signals for the search procedure at hand. These signals are then tested in reverse sequence from n to 1, and the signal at stage i is used either to retain the output at stage i or to eliminate it by shifting up one stage all current outputs from stages i+1 through n. If shifting occurs, the filler word is entered into stage n. A collection of shift registers is used for assembling the outputs from the merge network and for retaining the appropriate entries during the compression process. The

arrangement of these components is shown in Figure 2.6. In addition, the coordination network contains a collection of n registers not shown in the figure which serve as a buffer for data being transferred into memory.

On the left side of the figure are the circuits which generate the required control signals. Each of these has the following inputs:

RESET - initializes the circuit for a new cycle of operation.

M_i, $\overline{M_i}$ - $i^{\underline{th}}$ output from the merge network. Both the true signal and its complement are assumed to be available. M_i is used internally and is also passed directly to the output as x_i.

x_{i+1} - $i+1^{\underline{st}}$ output from the merge network.

A, O, N - control signals used to select the desired coordination function. Each of these signals is normally 1, and is changed to 0 when in active use.

C, C1 - timing signals.

$E1_{i+1}$ - a control signal from the next higher numbered stage.

$E2_{i-1}$ - a control signal from the next lower numbered stage.

REO - a two-way line used to broadcast the current instruction to all stages during the compression operation.

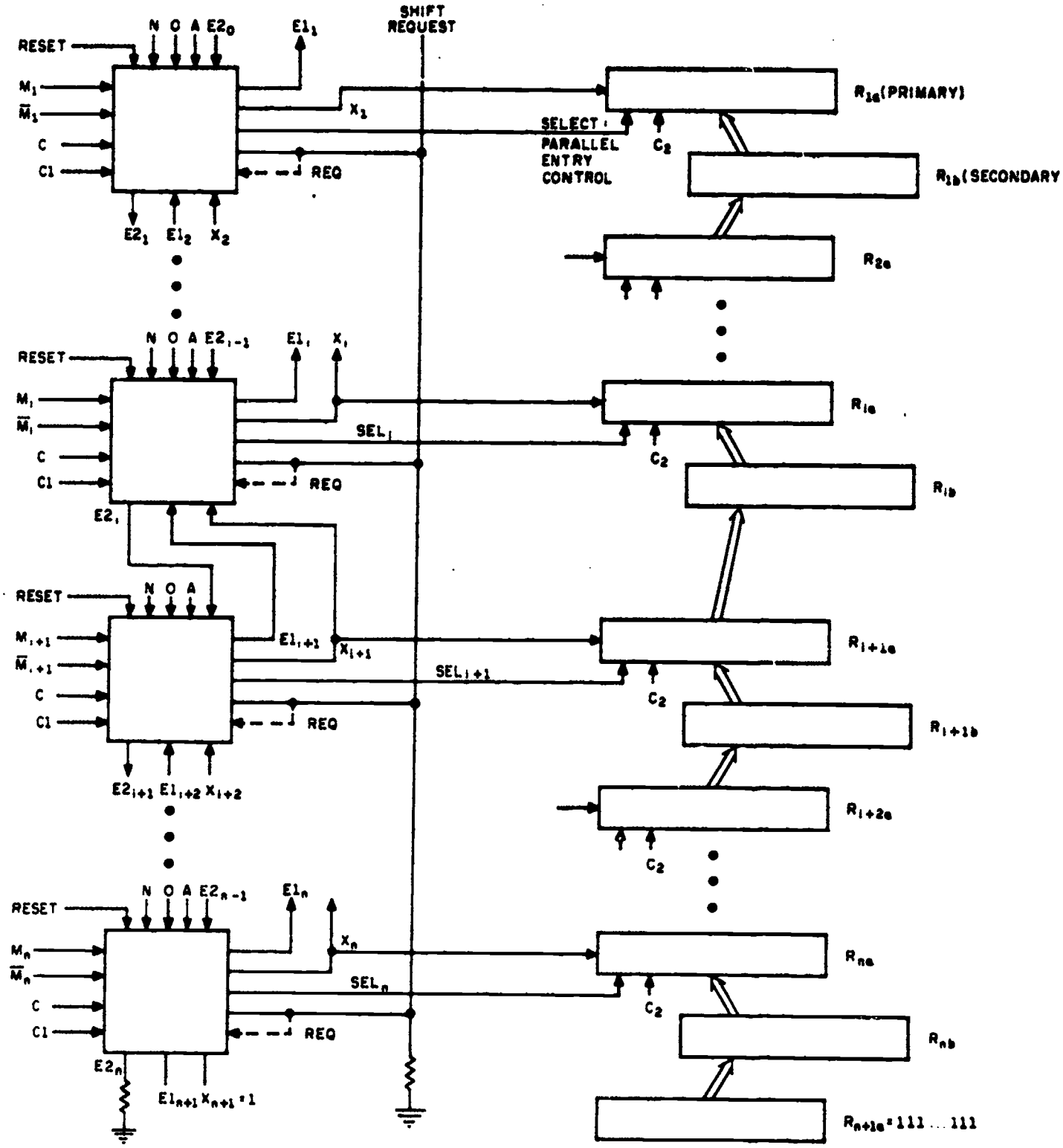


Figure 2.6. Coordination Network Interconnections

Outputs include $E1_i$ and $E2_i$ which are transmitted to the neighboring stages; and x_i, the 33-bit serial output from the merge network, which is collected in the primary register at stage i for further processing and also transmitted to the control cirtuit for stage i-1.[3] The SEL and REQ signals control shift register operation.

Each stage of the coordination network contains one primary and one secondary 32-bit processing register. The primary registers must be able to perform shifts (for collecting serial data) and have parallel input and output facilities. The secondary registers, which may be simpler devices than the primary registers, serve to isolate the primary registers and hold data temporarily on its way from one stage to another: no shifting capability is required. The secondary register in stage n should have all its input lines permanently set at 1.

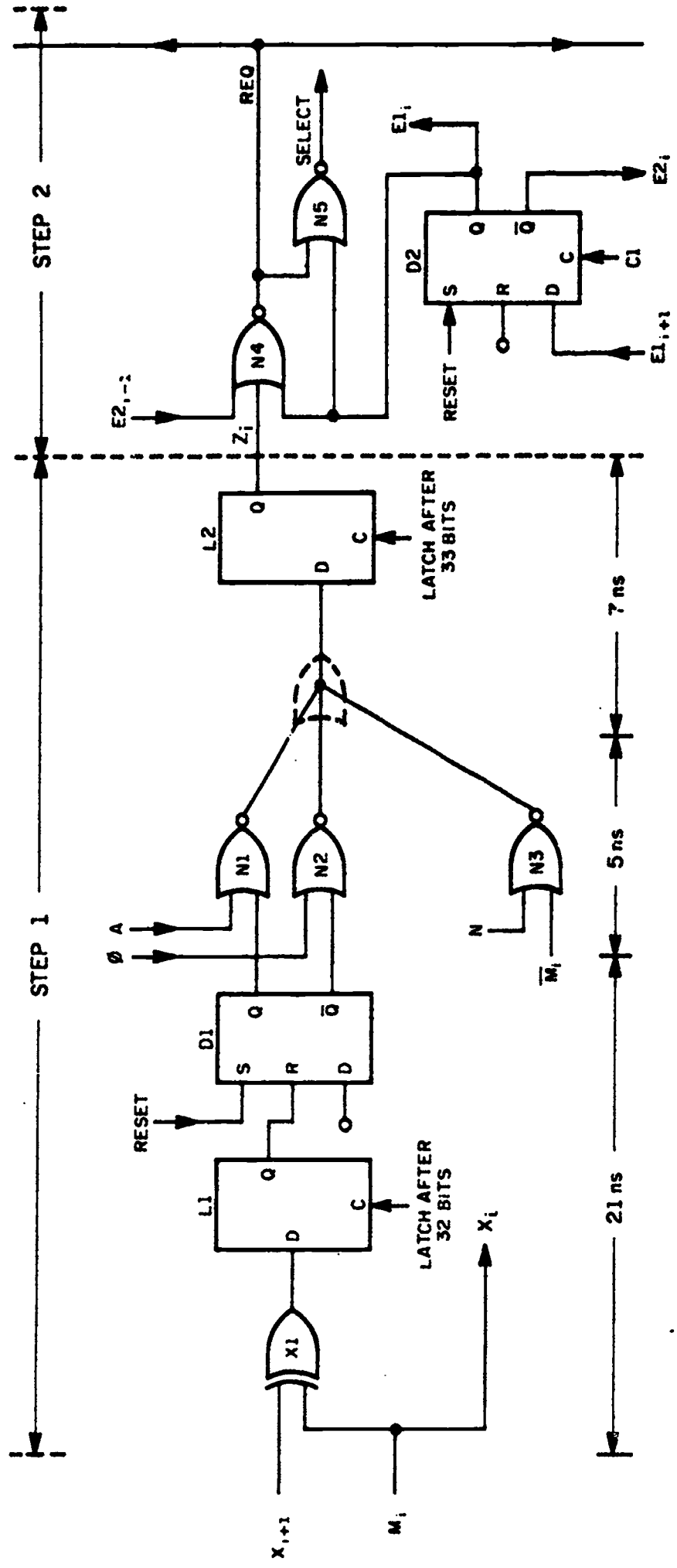
Operation of this network proceeds in three phases, to be referred to as steps 1, 2 and 3, where steps 1 and 2 can be implemented as shown in Figure 2.7.

2.3.2.2 Step 1 Processing

The stage i output of step 1 is the signal z_i, which must be available for all i before step 2 begins. The system has been designed so that no matter which procedure (AND, OR, NOT) is being performed the signal $z_i = 1$ causes the contents of primary register i to be "erased", while the signal $z_i = 0$ causes the contents of primary register i to be retained. The

---

[3]Only the 32-bit document identification number must be saved in the registers.



A-O-N SIGNAL LEVELS

PROCEDURE	A	O	N
AND	L	H	H
	H	H	L
OR	H	L	L
	L	L	L
NOT	H	L	
	L	H	

NOTE: INPUT CONSISTS OF 33 BITS PRESENTED SERIALLY AT M_i AND X_{i+1}. DISABLE CLOCK TO D1 AFTER 32nd BIT; LATCH L2 AFTER IT RESPONDS TO 33rd BIT.

Figure 2.7. Details of Coordination Steps 1 and 2

following rules determine z_i, where w_i is the complete document identification number associated with stage i:

For "A OR B", $z_i = 1$ if and only if $w_i = w_{i+1}$, (2.2)

For A AND B, $z_i = 1$ if and only if $w_i \neq w_{i+1}$ (2.3)

For A AND NOT B, $z_i = 1$ if and only if either (2.4)

$w_i = w_{i+1}$ or w_i originated

on list B.

The origin of word w_i is determined from the 33rd bit at stage i, which is 0 for items from list A and 1 for items from list B. This last bit is discarded after the necessary determination has been made.

As shown in Figure 2.7, the equality or inequality of w_i and w_{i+1} is determined by means of flip-flop D1 and latch L1. Output Q of the flip-flop is initially set to 1. As long as successive bits of w_i and w_{i+1} remain equal, the output of the exclusive OR (and of L1) is 0, and D1 does not change. However, as soon as any pair of bits from w_i and w_{i+1} fail to match, D1 is switched. Output Q then remains zero until another reset signal is received. The clock signal to latch L1 is normally kept high to prevent spurious signals from affecting D1. L1 is "opened" after each of the first 32 bits is received, but not after the 33rd bit. In this way the Q output of D1 is made to indicate whether or not the two adjacent document identification numbers are equal, but it is not affected by values of the source tags transmitted as the 33rd bit. The effect of source tags is registered by gate N3, whose output can be non-zero only when the "AND NOT" operation is being performed.

Finally, the control signal z_i is generated as the implied OR of the outputs from gates N1, N2 and N3 after transmission of the 33rd bit and retained at the output of latch L2 until after step 2 of the coordination processing is complete. When the coordination procedure is AND or OR, only gate N1 or N2 conducts, and z_i is formed according to (2.2) or (2.3). When the coordination procedure is NOT, both N2 and N3 conduct; and, as (2.4) requires, $z_i = 1$ whenever $w_i = w_{i+1}$ or whenever the 33rd bit transmitted at stage i is 1, i.e., when the i^{th} document identification number originated on list B.

The propagation time through step 1 depends upon the path of interest (Figure 2.7), but in any case it is always shorter than the propagation time through a comparison element in the merge network. Thus, the only delay of real interest is the 12ns propagation time of bit 33. For the sake of uniformity in hardware timing and operation, it is assumed that 33-bit inputs are always used and that signal z_i is available 12ns after the thirty-third input is received at the step 1 terminals. The 33rd bit, of course, has no effect on the value of z_i for AND and OR processing.

2.3.2.3 Step 2 Processing

The operation of coordination step 2 will be explained with the aid of Figure 2.8 and Table 2.4. Figure 2.8 is a logic diagram for the last three stages (n-2 through n) of step 2, illustrating the interconnections between stages. Table 2.4 lists the signal states in these stages during the first three cycles of operation.

The D2 flip-flops constitute a shift register which is used to

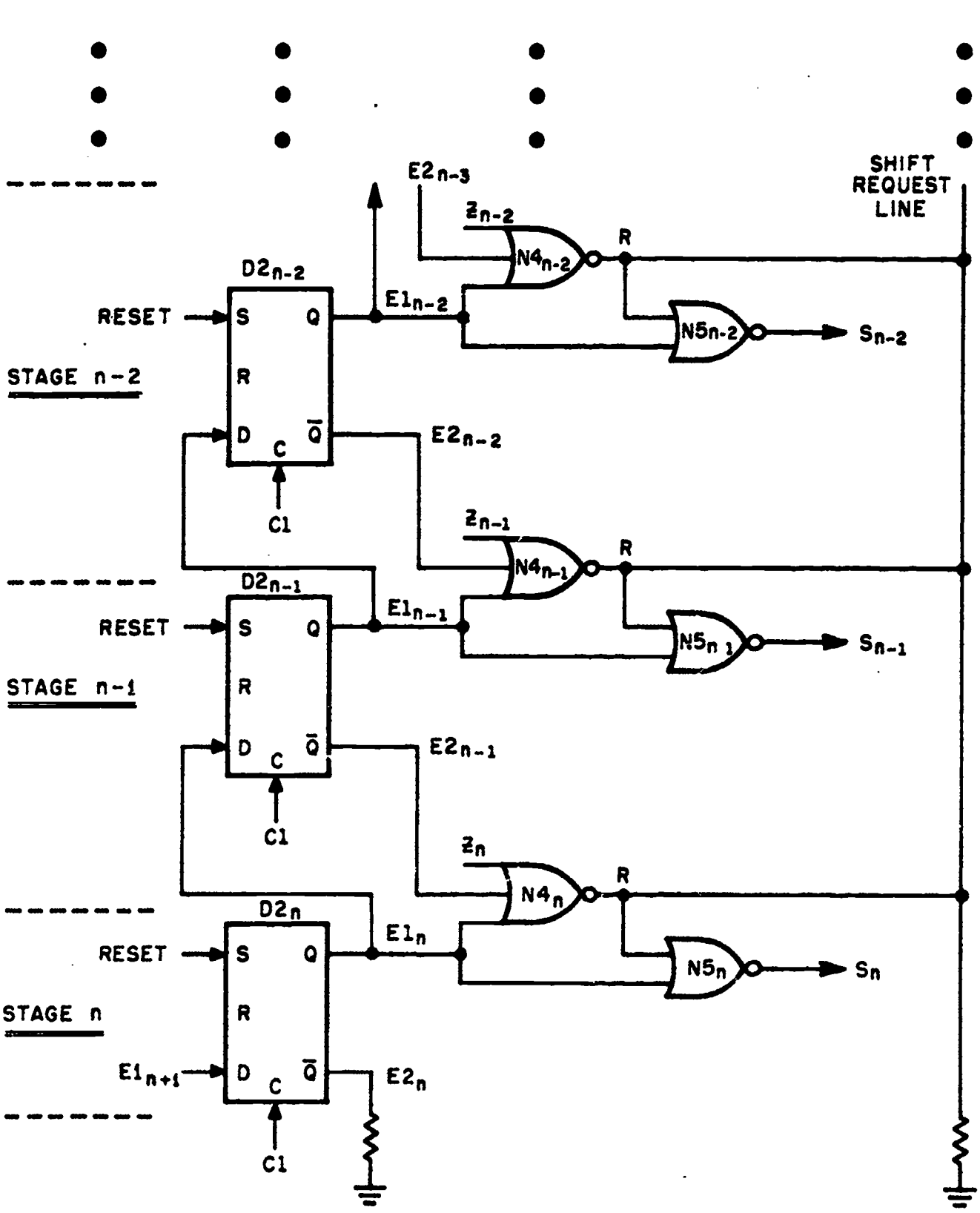


Figure 2.8. Coordination Step 2 Interconnections, Final Three Stages

	STAGE n+1	STAGE n				STAGE n-1				STAGE n-2			
	$E1_{n+1}$	$E1_n$	$E2_{n-1}$	R	S_n	$E1_{n-1}$	$E2_{n-2}$	R	S_{n-1}	$E1_{n-2}$	$E2_{n-3}$	R	S_{n-2}
INITIAL	1	1	0	0	0	1	0	0	0	1	0	0	0
CYCLE 1	0	0	0	$\overline{Z}_n$	Z_n	1	0	$\overline{Z}_n$	0	1	0	$\overline{Z}_n$	0
CYCLE 2	0	0	1	$\overline{Z}_{n-1}$	Z_{n-1}	0	1	$\overline{Z}_{n-1}$	Z_{n-1}	1	0	$\overline{Z}_{n-1}$	0
CYCLE 3	0	0	1	$\overline{Z}_{n-2}$	Z_{n-2}	0	1	$\overline{Z}_{n-2}$	Z_{n-2}	0	0	$\overline{Z}_{n-2}$	Z_{n-2}

Table 2.4. Coordination Step 2 Control Signals for First Three Cycles of Operation

control the selection of successive control signals. As Table 2.4 shows, the true outputs of all flip-flops (signals $E1_i$) are initially set to 1, causing R and all S_i to be 0. For the first cycle of operation, $E1_{n+1}$ is changed to 0 and C1 is pulsed, reversing the outputs of $D2_n$, but not those of any other flip-flop. At this point, $E1_n = E2_{n-1} = 0$, and $R = \overline{Z}_n$. Note that gate N4 is "off" in all stages except the $n^{\underline{th}}$ because the E1 signal in all other stages is 1. For the same reason, $S_i = 0$ for all $i \neq n$, but $S_n = Z_n$. Now a different clock signal, C2 (see Figure 2.6), causes each primary shift register whose SELECT signal (S_i) is 1 to accept inputs from the stage below.

During the second cycle of operation, the 0 at $E1_n$ is clocked through $D2_{n-1}$, reversing its state and leaving $E1_{n-1} = 0$ and $E2_{n-1} = 1$. Stage n-1 now behaves like stage n did in the previous cycle, setting $R = \overline{Z}_{n-1}$ and $S_{n-1} = Z_{n-1}$. At the same time, $E2_{n-1}$ turns off gate $N4_n$, isolating Z_n from the request line. $E1_n$, however, is still 0, and so $S_n = \overline{R} = Z_{n-1}$. All other stages are unaffected by these changes and generate $S_i = 0$. Now, if $Z_{n-1} = 1$, C2 will cause the primary register in each of the last two stages to accept inputs from the stage below. If $Z_{n-1} = 0$, however, no further action will occur during the second cycle.

Allowing a 5ns delay per gate, 7ns to switch D2 and 10ns to broadcast the REQUEST signal, $\overline{Z}_i$, the first phase of the step 2 operating cycle requires 27ns. During this same time period, under the control of C1, all secondary processing registers are loaded from the primary registers in the stage below. Six nanoseconds are required for loading primary registers

during the second phase of the cycle, giving a total cycle time of 33ns.

A coordination network with a 33 nanosecond operating cycle would require t_1 = 8.448 microseconds to process a list of 256 inputs. Of course it may be practical to bypass stages for which Z = 0, but 8.448 μ sec. remains as a worst case possibility.

Faster operation can be achieved by implementing step 2 as k smaller units which operate in parallel on input lists of length n/k and complete the task in t_1/k μ sec. Except, perhaps, for the duplication of control signals, no special provisions of any kind would be required to implement step 2 in this way--and problems associated with broadcasting the REQUEST signal, R, would be reduced. A small amount of additional complexity would be introduced into the control of step 3, where the outputs from the separate units would have to be combined into a single list.

2.3.2.4 Step 3 Processing

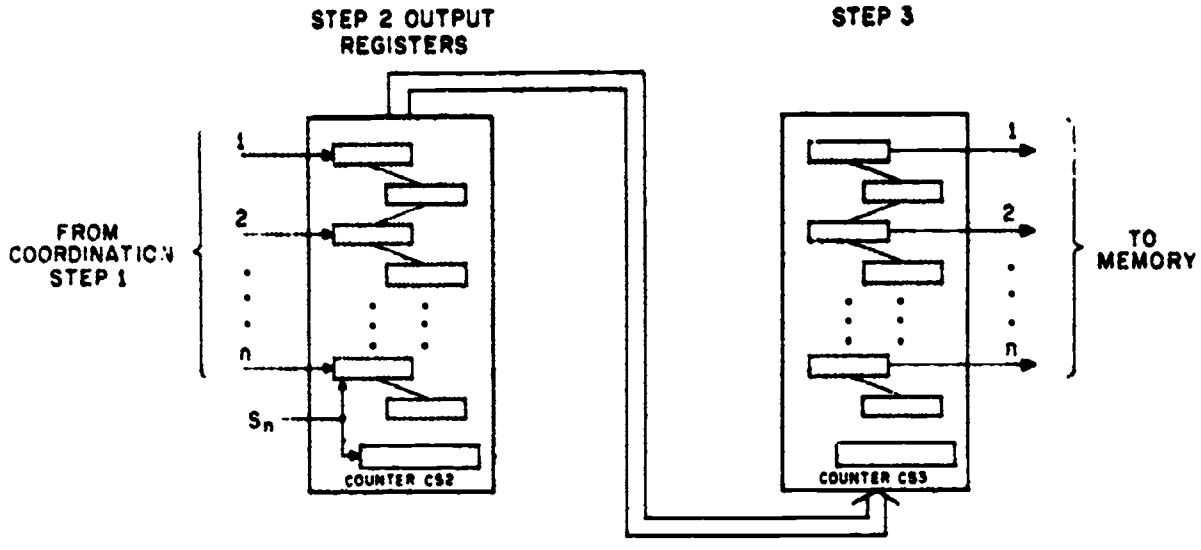
At the completion of step 2, the n primary registers in the step 2 processor contain some unpredictable number, k ($0 \leq k \leq n$), of valid document identifiers followed by n-k "fillers". These n words cannot simply be returned to memory because they may constitute only a small part of the output from the current coordination procedure, which may take several hardware cycles to complete. Retaining the output from step 2 after each cycle would produce a final result containing groups of valid pointers separated by groups of fillers. That would be unacceptable as input for further processing either as a part of the present search or in a subsequent search. Thus, it is necessary to collect valid results from step 2 until a complete sublist of n document identifiers is available or until the current process has been

completed. The last sublist returned to memory from any coordination procedure may, of course, contain fillers.

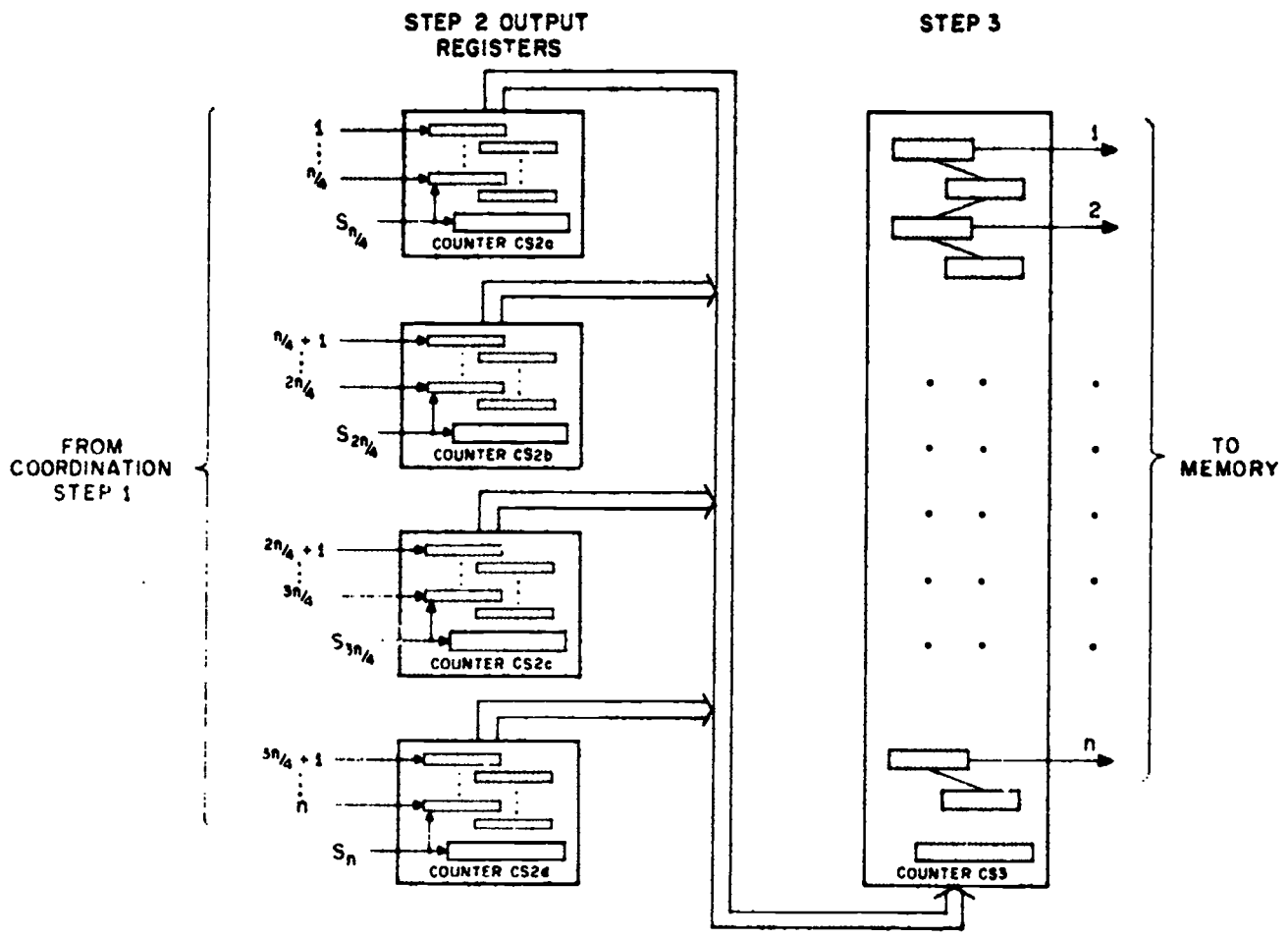
The required "packing" is accomplished by means of a second set of registers similar to those in the step 2 processor. Again a system of primary and secondary registers or equivalent physical devices is employed; and again the primary registers should be capable of serial shifting as well as parallel input and output, while the secondary registers need only perform parallel input and output operations. These registers serve both as a collection device for results from step 2 and as a transposer and buffer for results returning to memory. Their relationship to other parts of the system is shown in Figure 2.9.

This compression system is controlled by means of counters CS2 and CS3 in the step 2 and step 3 processing units, respectively. Before step 2 begins, CS2 is loaded with the number n. The $n^{\underline{th}}$ stage SELECT signal, S_n, and the C2 clock are used to decrement the counter each time the contents of the step 2 registers are shifted up one stage. When the procedure is complete, the first k registers contain valid results, the last (n-k) registers contain fillers, and the number k appears in CS2. This counter can then be used to control the number of shift cycles performed in moving the results into the step 3 unit. As Figure 2.9 indicates, data items move from the top of the list in step 2 to the bottom of the list in step 3.

CS3 is loaded initially with the value n and decremented each time an input is received from step 2. When the step 3 counter reaches 0, all step 3 registers contain valid results. At this time, further transfers from step 2 are suspended, the contents of the step 3 registers are returned to memory, CS3 is reinitialized and transfers are resumed. During all



(a) One Step 2 Processor



(b) Four Parallel Step 2 Processors

Figure 2.9. Transfer Mechanism Between Coordination Steps 2 and 3

hardware cycles except the last, transfers between steps 2 and 3 stop when CS2 reaches 0. In the last cycle only, CS2 and CS3 must both be decremented to 0 in order to eliminate unwanted entries from the top of the last sub-list and place fillers in their proper positions at the bottom.

If several step 2 processors operate in parallel, then step 3 control becomes slightly more complicated in that each of these units must be emptied in turn into the step 3 registers. The control unit will have to provide for the necessary switching and supervision.

An estimate of the time required in step 3 for a single move from one primary register to the next is 13ns, with 3.33 μs needed to transfer the entire contents of 256 step 2 registers.

2.3.2.5 Merge and Coordination Control Requirements

Control unit requirements for the hardware system described here are very modest. A 20 MHz (50ns pulse interval) clock and a signal indicating the availability of data from the memory are required to control the merge network, the latch in coordination step 1, and the shift register inputs in step 2. In addition, one or two counters are needed to initiate and terminate various step 1 operations at the proper times relative to the operation of the merge network.

Step 2 requires a timing interval of about 33ns (50 or 25ns might be acceptable) and a provision to produce a second clock pulse at a fixed interval relative to the first. This step generates internally a signal pair ($E1_1 = 0$, $E2_1 = 1$) which can be used to determine when processing is complete without any additional control unit activity. Step 3 requires a single clock for the primary registers and appropriate circuitry to delay

the clock signal for the secondary registers and to monitor the various counters and generate requests fo· memory transfers. A clock interval of 12.5ns is adequate for step 3 so that a basic clock frequency of 80MHz and the submultiples of 20 and possibly 40 MHz can be used to control the entire hardware system.

2.3.3 Data Memory

The proposed design requires a memory with a very high data rate ($O(10^9)$ bits/sec.) and short effective cycle time (100ns or less). While these requirements are stringent, they can presently be met either directly by means of bipolar devices or indirectly by interleaving slower MOS units.

Because blocks of information are routed through the system in a serial-by-bit, parallel-by-word fashion, it is natural to store data in "transposed" format. That is, the $i^{\underline{th}}$ n-bit physical word in the memory may actually contain the $i^{\underline{th}}$ bit from each of n data items. For the standard system under discussion k-word by 256-bit memory modules would be used, where k is an integral multiple of 32.

Under the most severe operating conditions, simultaneous input and output to both the disk and the hardware coordination system would be required, and high priority memory transactions would occur at the rate of about one every 100ns. With n = 256, the corresponding overall transfer rate would be approximately 2×10^9 bits/second. As shown in section 2.4 below, all required transfers can be accomplished simply and without serious conflicts using either a single 100ns cycle memory module or a collection of four interleaved submodules each with a cycle time of 400ns.

At least one semiconductor manufacturer, Intel [9], is now promoting a 100ns bipolar memory system with the required characteristics at a

price of about ten cents per bit. More such systems and lower prices are to be expected in the near future. A number of 400ns MOS units are also available.

2.3.4 Disk File System

To match the high speed parallel processing capabilities of the merge and coordination networks, a wideband mass storage device is required for the postings file. Analyses to date have assumed the use of a head-per-track disk with the capability of transmitting n tracks simultaneously. The system should be capable of reading n tracks from one channel while writing n tracks on another. It should also be equipped with a hardware queuer which would permit the servicing of a group of outstanding I/O requests in the order in which the referenced addresses became available, reducing considerably the time required to process search requests involving large numbers of terms.

The Illiac IV disk file system [10-12], effectively meets all these requirements. It consists of two Burroughs Model II disk files operating on separate channels, each with sufficient electronic circuitry for reading or writing simultaneously on 128 tracks of one disk. Both channels can operate concurrently at full capacity. The system employs a disk file optimizer (hardware queuer) which accommodates up to 24 outstanding I/O requests. The total storage capacity of this system is 10^9 bits, and its maximum transmission rate is 10^9 bits/second.

2.3.5 Control Computer

At the present stage of design, no specific implementation can be given for the control computer. Conceptually, it is a computer with

responsibility for a number of supervisory functions to be performed before or during the operation of the specialized hardware. Some of these functions may be distributed among the controllers for the individual devices, they may be performed by one or more dedicated processors which have been optimized for this application, or they may reside mainly within the computer which has overall control of the retrieval system.

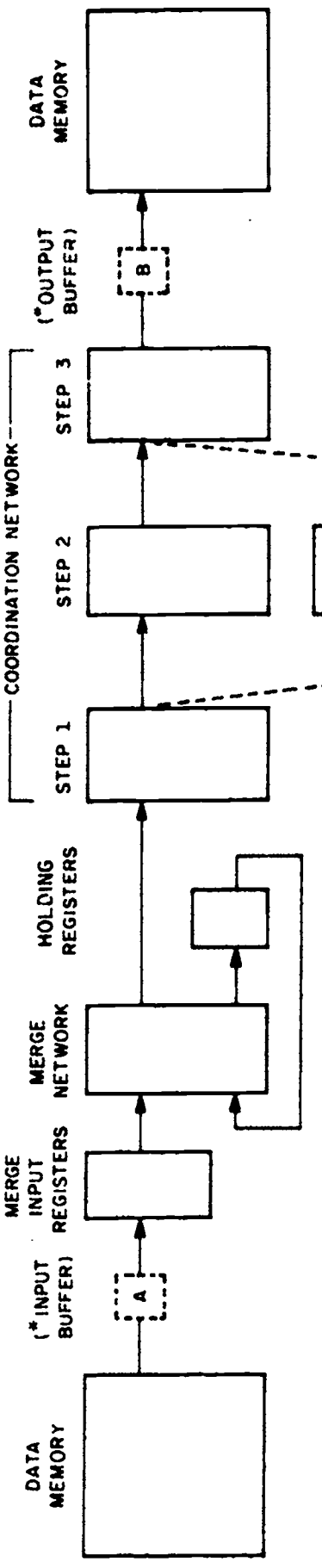
The required functions are of four broad types: communication with the rest of the system, memory management, routing of sublists, and internal control. The communication function consists of accepting requests for service and postings file addresses from the main system and generating the appropriate signals and control information when a search is complete. Memory management refers to the dynamic allocation of space in the data memory and on any scratch disks which may be used to process a search. This function represents a heavy computational load since, during any hardware cycle, two sublists may be removed from the data memory and two more may enter. Routing in the present context means providing sublists to the merge network (and to the disk) in the proper order (see section 3.1). Finally, internal control refers to algorithmic decisions such as when to write intermediate results on disk and whether to read, merge or skip a particular list when it becomes available. These decisions are dictated by the system resources available and the nature of the search at hand. They have a crucial role in determining the overall performance capabilities of the system.

2.4 System Integration

The purpose of this section is to illustrate how the various subsystems function together, especially with respect to their separate timing

requirements and to the total time available for a complete operational cycle. Without presenting an exhaustive catalog of possible designs, the range of alternatives available is outlined. The approach here is to choose a basic design which satisfies all constraints and then examine various departures from this design which may be desirable and various problems which can arise. It is assumed that the data memory contains a number of independent modules which may be accessed simultaneously. The term memory conflict implies multiple simultaneous requests for access to a single module: only one such request can be honored during any given memory cycle. Multiple requests involving different modules may be serviced concurrently and do not present conflicts. Timing information from sections 2.3.1 and 2.3.2 will be used extensively, and it may be helpful to refer to Figure 2.7.

First, consider the overall organization of the system and the timing requirements for individual operations, as shown in Figure 2.10. For the basic design analysis, assume that n, the number of parallel data paths, equals 256. A hardware cycle begins with 32 memory fetches to obtain data. The rate at which this information may be applied to the hardware is determined by the cycle rate of the memory, subject to the 42ns minimum interval between bits required by the merge network. A thirty-third bit supplied by the control system propagates through the merge network in 378ns, and requires an additional 12ns in step 1 of the coordination system. If step 2 is implemented as a single unit with 256 registers, then it requires a processing time of 8.448 μs. If a cluster of four identical 64-register units operating in parallel is employed, then the processing time can be reduced to 2.112 μs. Coordination step 3 contains 256 primary



MEMORY	MERGE	STEP 1	COORDINATION STEP 2	STEP 3	MEMORY
0.050-0.200 µs per cycle for 32 cycles (one word contains 32 bits)	0.378 µs total delay for one bit 0.042 µs minimum interval between bits	0.012 µs delay for bit 33 ≤ 0.033 µs delay for bits 1-32 (masked by merge operation)	0.033 µs processing per word For 256 words: 8.448 µs in one unit; 2.112 µs in four parallel units	0.013 µs processing per word For 256 words: 3.328 µs	0.050-0.200 µs per cycle for 32 cycles (one word contains 32 bits)

*Buffers A and B may be inserted for control of memory conflicts.

Figure 2.10. Timing Summary for Hardware Subsystems

registers and requires up to 3.328 μs for its operation. Finally, 32 memory cycles are required to transfer results back into memory. Cumulative processing time requirements, assuming no memory conflicts, are shown in Table 2.5 for systems with effective memory cycle times of 50, 100 and 200ns and with 1 and 4 step 2 processing units. Table 2.6 contains the corresponding data for a system with n = 16, except that only one step 2 processor is considered.

The time required for the disk to read one word from each of n tracks is approximately 14 μs (13.89 μs), and this determines the maximum allowable processing time for one hardware cycle. Two candidates from Table 2.5 meet this criterion. Both employ four parallel step 2 processors; one has a 50ns memory cycle, and the other has a 100ns cycle. The 100ns system is chosen as a standard, and the remainder of this chapter is devoted to a further analysis of its timing requirements.

A time distribution for processing activities in a typical operating cycle of the standard system is shown in Figure 2.11. In the absence of memory conflicts, the entire procedure including the return of results to memory can be completed within 12.33 μs, leaving about 10% of the available time (region f in the figure) free for contingencies. Times shown include generous allowances for delays within the circuit components, and they reflect the worst-case situation in which two, 256-input lists, without any common entries are processed using the operation OR. Duplication of elements tends to reduce the size of region d. For AND's, which normally produce only a few result postings, region d is usually much shorter and region e frequently disappears altogether as the results of several hardware cycles may be collected in the step 3 output buffer.

EFFECTIVE MEMORY CYCLE	†MEMORY ACCESS COMPLETE (32 CYCLES, NO CONFLICTS)	*MERGE START (BIT 1 IN)	MERGE COMPLETE (BIT 33 OUT)	*COORD. STEP 1 COMPLETE (BIT 33 OUT)	NO. OF PARALLEL STEP 2 UNITS	COORD. STEP 2 COMPLETE	**COORD. STEP 3 COMPLETE (256 TRANSFERS)	**MEMORY WRITE COMPLETE (32 CYCLES, NO CONFLICTS)
50.ns	1.600 μs	0.050 μs	2.028 μs	2.040 μs	1	10.488 μs	13.816 μs	15.416 μs
50.	1.600	0.050	2.028	2.040	4	4.152	7.480	9.080
100.	3.200	0.100	3.678	3.690	1	12.138	15.466	18.666
100.	3.200	0.100	3.678	3.690	4	5.802	9.130	12.330
200.	6.400	0.200	6.978	6.990	1	15.438	18.766	25.166
200.	6.400	0.200	6.978	6.990	4	9.102	12.430	18.830

Conditions:

Memory Module Dimensions: k words × 256 bits

256 Parallel Data Paths

No Memory Conflicts

---

*Three processes proceed simultaneously: input from memory, merge, and coordination step 1.

**During any particular hardware cycle, memory output may or may not take place. If present, it may follow coordination step 3, or it may interrupt step 3.

Table 2.5. Cumulative Timing for Hardware Cycle with 256 Parallel Data Paths

EFFECTIVE MEMORY CYCLE	*MEMORY ACCESS COMPLETE (32 CYCLES, NO CONFLICTS)	*MERGE STAPT (BIT 1 IN)	MERGE COMPLETE (BIT 33 OUT)	*COORD. STEP 1 COMPLETE (BIT 33 OUT)	NO. OF PARALLEL STEP 2 UNITS	COORD. STEP 2 COMPLETE	**COORD. STEP 3 COMPLETE (16 TRANSFERS)	**MEMORY WRITE COMPLETE (32 CYCLES, NO CONFLICTS)
50.ns	1.600 μs	0.050 μs	1.860 μs	1.980 μs	1	2.508 μs	2.716 μs	4.316 μs
100.	3.200	0.100	3.510	3.522	1	4.050	4.258	7.458
200.	6.400	0.200	6.810	6.822	1	7.350	7.508	13.908
300.	9.600	0.300	10.110	10.122	1	10.650	10.858	20.458
400.	12.800	0.400	13.410	13.422	1	13.950	14.158	26.958

Conditions:

Memory Module Dimensions: k words × 16 bits
16 Parallel Data Paths
No Memory Conflicts

---

*Three processes proceed simultaneously: input from memory, merge, and coordination step 1.

**During any particular hardware cycle, memory output may or may not take place. If present, it may follow coordination step 3, or it may interrupt step 3.

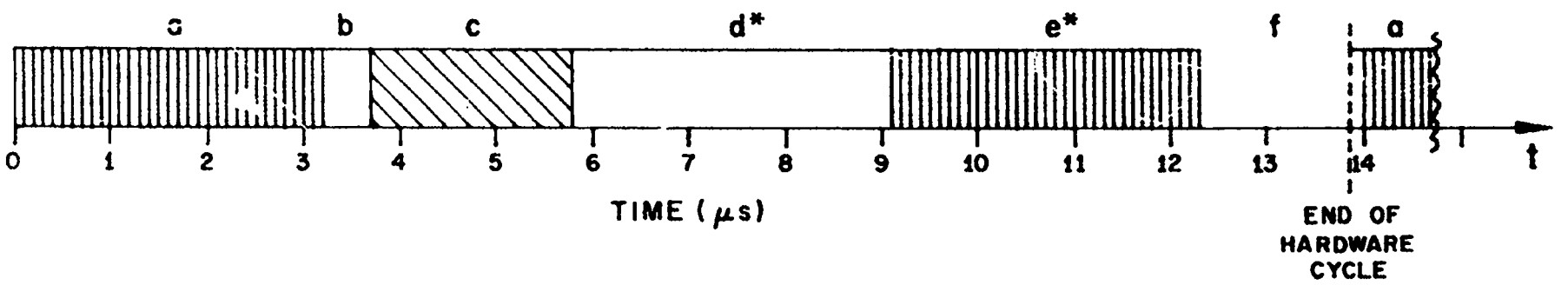
Table 2.6. Cumulative Timing for Hardware Cycle with 16 Parallel Data Paths

Conditions:

Parallel data paths: 256

Effective memory cycles: 100ns

Step 2 processors: 4



Activities:

a) Memory fetch, merge, and coordination step 1 processing

b) Completion of merge and coordination step 1

c) Coordination step 2

d) Coordination step 3

e) Memory store

f) Idle

*During any particular hardware cycle, memory output may or may not occur. If present, it may follow coordination step 3 or it may interrupt step 3.

Figure 2.11. Time Distribution for Hardware Activities in the Standard System

Finally, it is important to note that memory activity associated with a hardware cycle is concentrated near the beginning and the end of that cycle. Therefore, as long as the 14 μs time constraint is satisfied, no memory conflicts between phases a and e can ever occur. The only conflicts which may arise involve either phase a or phase e and disk I/O.

Figure 2.11 is based on clock intervals of 100ns for merge and step 1, 33ns for step 2 and 13ns for step 3. Use of an 80 MHz clock with its related frequencies as described in section 2.3.2.5 would increase the total time for a hardware cycle to 13.29 μs, which is still within the 14 μs limit.

So far in this analysis conflicts among the four groups of memory transfers which take place during a hardware cycle have been ignored. Figure 2.12 illustrates certain conflict situations and methods for controlling them. Each time line in the figure represents one hardware cycle (13.89 μs), and each vertical spike marks the beginning of one, 100ns memory cycle used for the indicated series of transfers. Processing times are based on data for the "standard" system in Figure 2.10 and Table 2.5. The reference line at the bottom of Figure 2.12 represents disk I/O requirements for all cases and is to be used separately with each of the other lines: a, b, c and d.

Figure 2.12(a) presents the same no-conflict situation as Figure 2.11. Disk accesses are spread uniformly throughout the cycle, and hardware transfers are grouped together near the beginning and end. While hardware input and output conflicts cannot occur if a strict 13.89 μs time limit is observed, other types of conflicts are nearly inevitable since, for example, data being read from disk and data being transferred to hardware are typically different sublists of the same postings file and therefore should

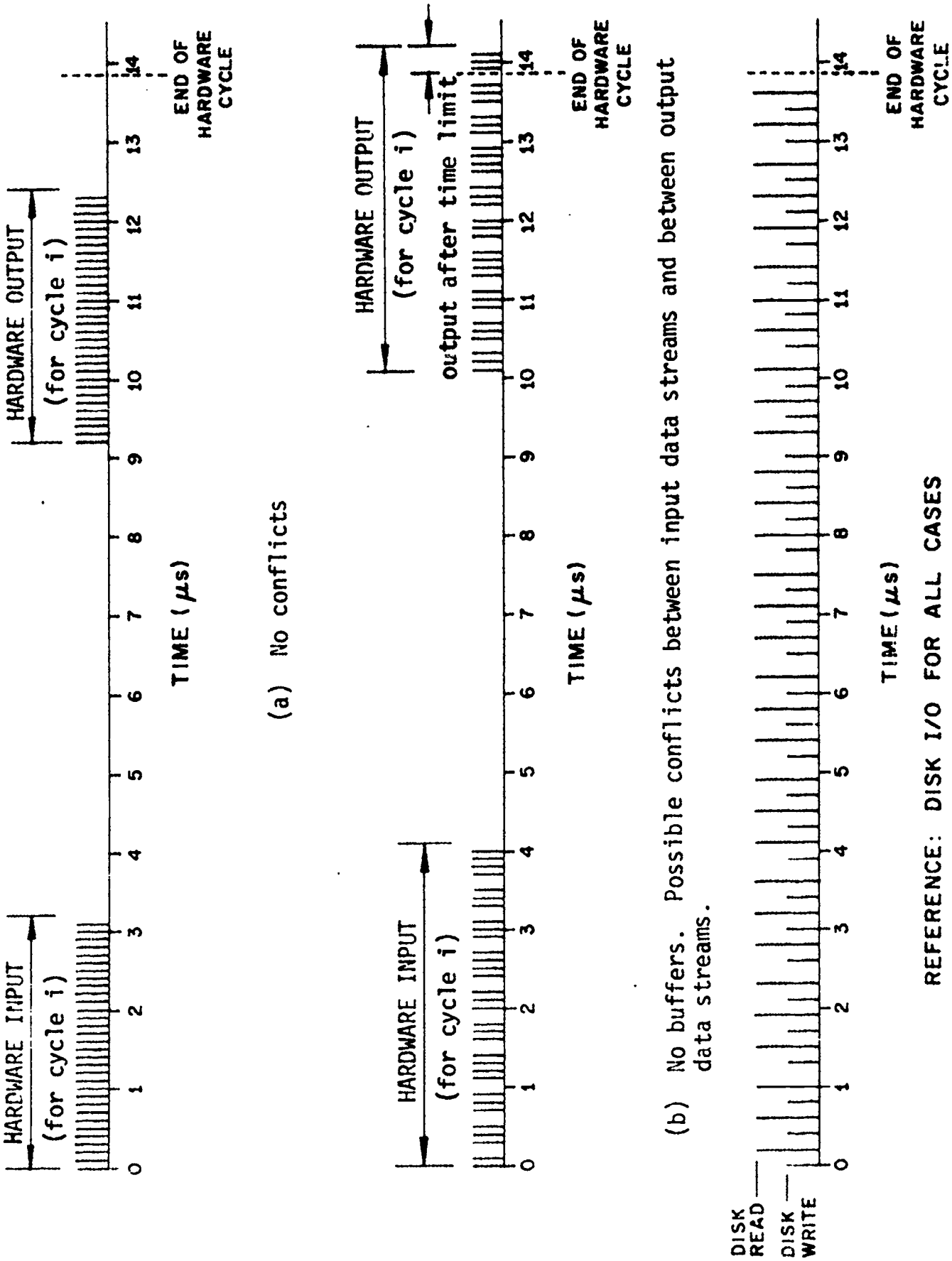


Figure 2.12. Memory Conflict Analysis

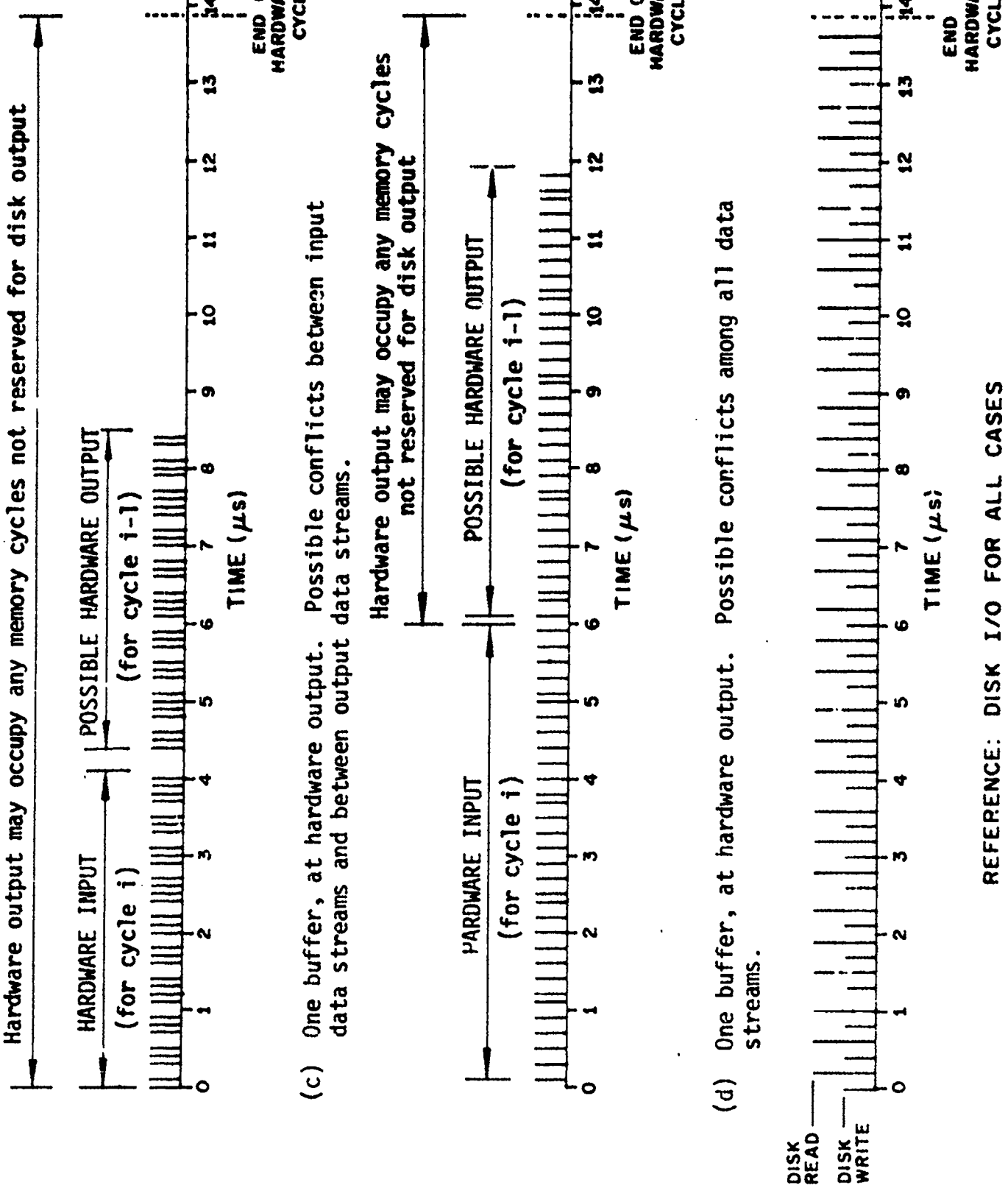


Figure 2.12 (continued). Memory Conflict Analysis

require access to the same memory module. A similar relationship exists between data transfers from hardware to memory and from memory to disk.

Suppose that the two input data streams do require access to the same memory module, i, and that the two output data streams require access to some other module, j. This case is illustrated in Figure 3.12(b). Here, disk transactions "steal" a number of memory cycles from the hardware input process, delaying its completion by approximately 1 μs. This delays completion of the hardware processing by the same amount; and another series of conflicts in the output module causes the total time required for the process to be 14.2 μs, about 300ns more than the allowable time.

One solution for this problem is to insert buffers between the memory and the hardware (points A or B or both in Figure 2.10) to spread memory access requirements more evenly throughout the operational cycle. Each buffer used in this way increases the total time required to process two lists by one hardware cycle, since, with a buffered input, each input sublist reaches the merge network one cycle later than before, and similarly at the output. Simulation experiments (Chapter 4) show that for a complicated sample search, the use of buffers has very little effect upon performance and, for any given set of starting conditions, may either increase or decrease the total time required. When buffers are used, Equation 2.1 becomes

$$t = i+j+w + 1+N_B \ , \qquad (2.5)$$

where N_B is the number of buffers employed in the hardware path.

Returning to the conflict situation defined above in which two input data streams share one memory module and two output data streams share

another, suppose that data from coordination step 3 were collected in a buffer at Point B (Figure 2.10) and returned to memory at any convenient time during the next hardware cycle. The effect would be to distribute memory access requirements for hardware output over an entire cycle instead of concentrating them near the end. As Figure 2.12(c) shows, all timing requirements can be satisfied easily using this configuration.

Figure 2.12(d) illustrates the situation in which all data transfers reference a single 100ns memory module. About half the memory cycles are required for disk I/O; and, as a result, the hardware input phase requires 6 μs. During the next 5.9 μs disk I/O continues, and the hardware output from the previous cycle is returned to memory. The entire cycle of operation including all hardware-related memory transactions can be completed in 11.9 μs. Again, only one buffer located at Point B of Figure 2.10 is needed.

If, instead of using a single memory module with a true 100ns cycle, an effective 100ns cycle were achieved by interleaving four cheaper 400ns submodules, then it can be shown that it would still be possible to satisfy the timing constraints with the aid of buffers at Points A and B of Figure 2.10 and with very little performance degradation.

2.5 Summary

This chapter has described in detail the hardware requirements of the proposed system. The operation of each of the hardware subsystems has been defined and designs have been outlined for a comparison element (the basic building block of the merge network) and for the various parts of the coordination network. Other subsystems can be obtained from existing devices either directly or through relatively minor modifications. Timing

constraints have been analyzed, and the interaction of the various components during a typical cycle of operation has been discussed at length. The effects and control of memory conflicts have also been evaluated.

It is concluded that hardware term coordination systems capable of processing up to 256 items simultaneously can reasonably be built using current technology.

3. BASIC ALGORITHMS

In order to operate the proposed hardware coordination system, a number of procedural decisions are required. The system is intended for operation in a large, on-line retrieval environment where frequently a single search request may involve a large number of search terms and hence require the manipulation of a large number of postings files. Further, the number of entries in these files may vary radically from one file to another and may be expected frequently to exceed the number of data paths in the system and even the available memory. In the data base used as a model for this study, for example, some terms index as few as one or two documents while others index as many as half a million. As a result, procedures are needed for insuring a proper sequence of inputs to the merge network, for handling intermediate results in large searches, and for processing excessively long lists. Problems of this type are considered in the present chapter, and a brief description of the standard algorithms which have been adopted for performance evaluation studies is presented. Variations examined in the interest of optimizing performance are discussed in Chapter 4. It is beyond the scope of this report to specify explicit algorithms for use by the control computer. Rather, it is assumed that the processing which must be done there can be performed within the available time.

3.1 Sublist Sequencing

Consider the problem of processing two lists, each of which contains more than n postings, where n is the number of data paths in the system. Each of the two input lists may be divided into sublists of length n, and one new sublist may be processed each cycle. The problem then becomes one of choosing

a proper sequence of sublists to assure the success of the overall merge. Clearly, some sequences are not appropriate since, for example, one could not normally process first all the sublists from one file and then all the sublists from the other. During each hardware cycle, n new inputs are introduced into the merge and the n smallest elements currently in the system are released as finished results and become unavailable for further sorting.

Refer now to Figure 3.1, where Lists 1 and 2 represent files to be merged. Items in each file are assumed to be arranged in nondecreasing order. Let α be the last n-element sublist processed from List 1, β the last sublist from List 2, and γ and δ the next sublists available on Lists 1 and 2, respectively. The last elements on α, β, γ and δ are a, b, e and f, respectively; and c and d represent the leading items on lists γ and δ.

Define: N^{k+1} = a list of n new inputs to the merge for cycle k+1.

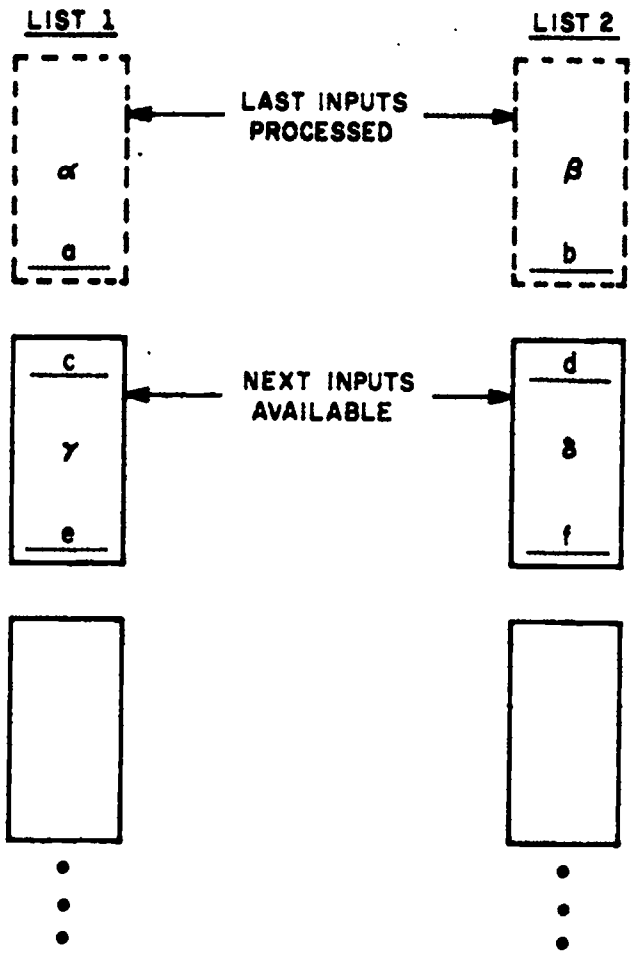
F^k = a list of n finished results, f_i, from merge cycle k

$(f_i^k \leq f_{i+1}^k, 1 < i < n - 1)$

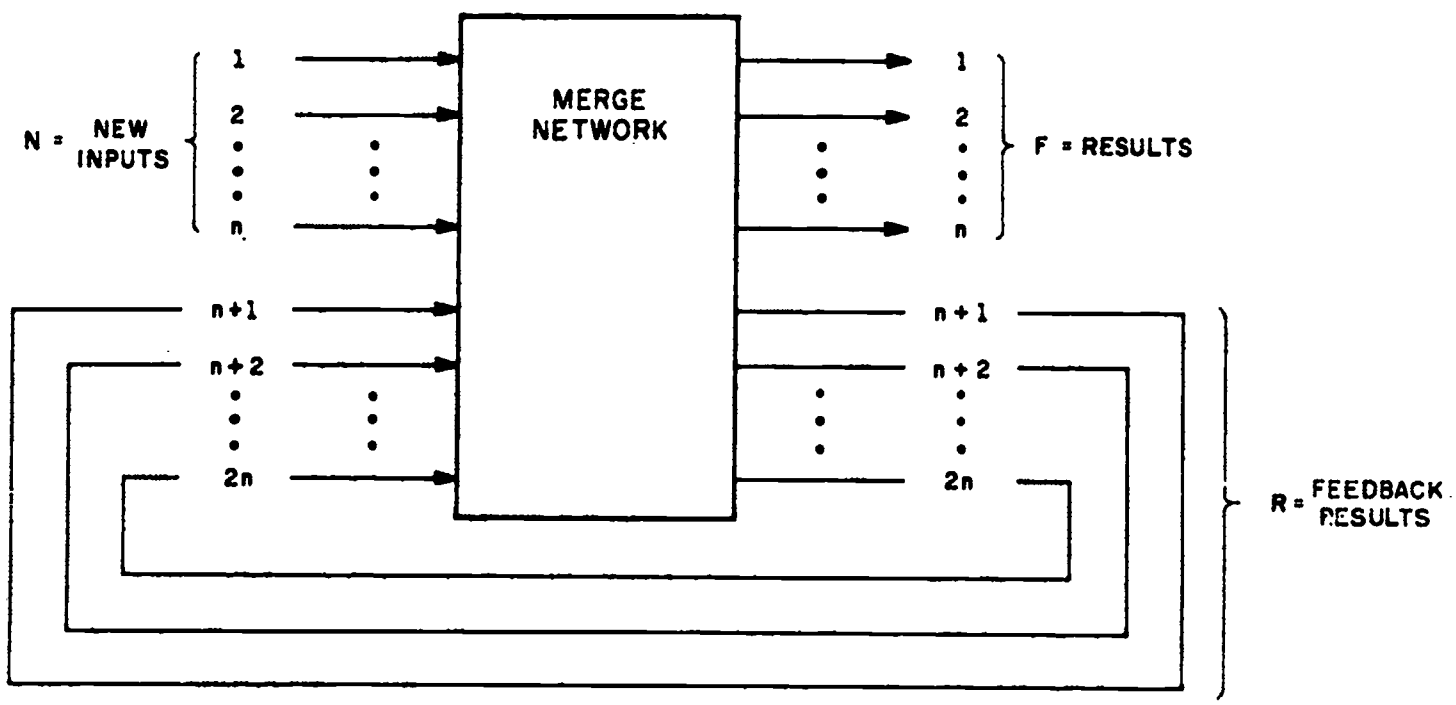
R^k = a list of n elements, r_i, retained for further processing after merge cycle k

$(r_i^k \leq r_{i+1}^k, 1 \leq i \leq n - 1)$.

Theorem: Proper sequencing of sublists is assured if, for every hardware cycle, the next available sublist having the smaller leading element is chosen. If the two leading elements are equal, either sublist may be used.



(a) Sublists and Data Elements



(b) Merge Network Inputs and Outputs

Figure 3.1. Definitions for Sublist Sequence Discussion

Proof: Consider the k^{th} step of the merge.

R^{k-1} merged with γ or $\delta \rightarrow F^k + R^k$

The last element of R^{k-1} is $r_n^{k-1} = \max R^{k-1} = a$ or b, say a.

Then $a \geq b$. Note also that $c \geq a$ since List 1 is arranged in nondescending order.

If $d \geq c$, then $d \geq a$ and

R^{k-1} merged with $\gamma \rightarrow F^k + \gamma$, where $F^k = R^{k-1}$ and $R^k = \gamma$

and R^{k-1} merged with $\delta \rightarrow F^k + \delta$, where $F^k = R^{k-1}$ and $R^k = \delta$.

If, however, $d < c$, then the relationship between a and d is unknown and R^{k-1} must be merged with δ so that any $\delta_i \leq a$ may be included in F^k. □

An alternative rule which will produce an acceptable sequence and which may be more convenient to apply in practice is based on comparison of a and b rather than c and d. As each new sublist is entered into the system, compare its largest (last) element with the largest item currently in the system and update an indicator showing which file has given rise to the current largest element. Then choose the next sublist from the other file.

3.2 Intermediate Results

When the length of the result of a particular search exceeds the available space in memory a series of intermediate results must be stored temporarily for later processing. In a given search, many but not all of these intermediate results tend to be of comparable lengths (greater than one memory load). It might seem appropriate to generate several such runs on one pass, combine them in pairwise fashion on the next pass beginning with whichever list becomes available first, and proceed in this fashion

until only one list remains. However, it has been found that this procedure results in a large amount of idle time spent waiting for the disk. Furthermore, if the number of intermediate results to be processed is odd, care must be taken to avoid an "infinite loop" situation in which any particular list serves alternately as a source on one rotation and a sink on the next. (In the example of the previous chapter, L1 was a source and L2 was a sink.)

It has proved more effective to identify the longest list at the beginning of a search and use it exclusively as a sink. Whenever the memory fills above a certain threshold, processing is suspended until the longest list becomes available, the contents of the memory are processed against the longest list and the result is left on disk. Then normal processing is resumed until the memory is full again.

3.3 List Splitting

It frequently becomes necessary to split a list into two sections, read the first part, and leave the other for future use. This facility is essential whenever it is necessary to process a source list which is too large to fit the available memory; it may be used (sparingly) at other times to improve performance by improving the utilization of the data memory. The procedure can be implemented as a simple bookkeeping transaction in the control computer.

3.4 Special Requirements of OR, AND and NOT Processing

Most of the discussion up to this point has dealt explicitly or implicitly with "OR" processing. From an operational point of view, no substantial difference exists among the OR, AND and NOT procedures, but certain details should be examined. For the remainder of this discussion

let L1 refer either to search term one or to its associated postings file, and let ℓ_1 be the number of document postings in that file. Let L_2, ℓ_2, LR and ℓ_r have corresponding definitions with respect to search term two and the result of the coordination procedure at hand. Assume that $\ell_2 \leq \ell_1$.

For the search request

L1 OR L2 ,

$$\ell_1 \leq \ell_r \leq \ell_1 + \ell_2 .$$

The condition $\ell_r = \ell_1 + \ell_2$ implies that no documents are common to both input lists, and processing proceeds exactly as described in Chapter 2. However if $\ell_r < \ell_1 + \ell_2$ then the smooth flow of results from the coordination network through the memory and onto the disk will be interrupted from time to time as it becomes necessary to wait for complete n-element sublists of LR. If the results are destined for memory alone, this delay presents no difficulty; but if they are to be written on disk, then "gaps" will appear in the disk files. The problem can be controlled by storing information on the disk to indicate which blocks contain valid data, or by supplying appropriate accounting procedures in the control computer. It can be eliminated by providing sufficient buffer space in memory to contain one complete logical track (n physical tracks) of information. In practical retrieval systems, the degree of overlap between any two pairs of postings files is believed typically to be quite small, perhaps 2%, so that in most searches only a few gaps might develop and a very small buffer would provide complete protection. In the worst possible case ("L1 OR L1") the density of information in the output file cannot drop below 1/2 its normal value since the result must contain at least ℓ_1 postings ($\ell_2 \leq \ell_1 \leq \ell_r$). Gaps in one

intermediate result may propagate to another, but this need not necessarily occur.

For the two search requests

L1 AND L2,

and

L1 AND NOT L2,

the problem of gaps on the disk need never arise since LR is never longer than L1 and hence the results of the search can be collected in memory until the procedure is complete. If the search involves very long input files, however, it may be necessary for the control computer to conduct these searches in several phases. Consider the search "L1 AND L2" in which $\ell_2 \leq \ell_1$, but ℓ_2 still contains km postings, where m represents the available memory space. L2 must be divided into k sections, $L2_1$, $L2_2$, ..., $L2_k$, each of the length m. The search may then be conducted in k+1 phases to form the desired LR:

$$LR_1 = L2_1 \text{ AND } L1$$

$$LR_2 = L2_2 \text{ AND } L1$$

$$\vdots$$

$$LR_k = L2_k \text{ AND } L1$$

$$LR = LR_1 \text{ OR } LR_2 \text{ OR } \ldots \text{ OR } LR_k .$$

A similar procedure is required to perform the search "L1 AND NOT L2" when L1 is too long for the available space.

3.5 Processing Algorithms for the Experimental System

For experimental purposes, a number of search simulations have been performed using a collection of standard procedures and parameters. This section describes these standard elements; Chapter 4 describes specific test conditions and presents results. As in other parts of this report, the term "merge" will often be used to refer to the complete hardware merge and coordination procedure.

3.5.1 Overview

Consider a search request involving the disjunction of several terms, and let the longest of the associated postings files be designated File S. Processing begins as soon as the disk addresses of the required files are determined. With the exception of File S, postings lists are accumulated in memory as they are encountered on the disk; merging is initiated whenever two lists are available and the merge system is free. When free memory drops below a specified threshold, t_1, further accumulation is suspended until some core is released or until the present contents are fully processed, coordinated with File S, and left on disk. Then normal processing is resumed.

3.5.2 List Selection

When a list other than File S is encountered on the disk it may be read into core, rejected or split. Normally it will be read in its entirety. A list may be rejected only when the required transmission facilities are busy (e.g., another list is being read) or when memory is

filled above the threshold level of (100-t)%.[1] A list rejected on one rotation is reconsidered on each succeeding rotation until it is finally processed. If at least t% of the total memory is free but the new list is still too long to fit the available space, then the list is split into two sections, A and B. Part A, which just fills the available space, is read immediately; the remainder of the list, Part B, is left for another rotation.

3.5.3 Merge Initiation

Merging is initiated whenever the merge system is free and any two lists are available. A list is considered available when either a) it is completely contained in core, or b) its first block is encountered on the disk. A list in the process of transmission from disk to core does not become available until after that transmission is complete. If a choice exists among more than two lists, the two shortest are selected for processing. Thus an attempt is made on a local basis to optimize the merge and coordination procedure. It can be shown [13,14] that merge time would be minimized if all lists were available initially and if the shortest two remaining lists were chosen for each new processing cycle. In the present context, minimizing the use of the merge system is not equivalent to minimizing the total elapsed time for a search; nevertheless, a strong interdependence between the two has been observed.

---

[1] One additional restriction in the present implementation can cause rejection: no more than 20 files for any given search may exist in core at one time. This limit is occasionally reached.

3.5.4 File S Processing

The longest file in a search is designated from the beginning as the sink and is used to collect and coordinate intermediate results. This list is accepted for processing only when no other lists remain on disk or when the memory is nearly full and must be cleared to make room for other files.

Certain other conditions must also be satisfied before File S can be processed, namely, all the required transmission facilities (input and output channels) must be free, the merge system must be idle, and adequate space must be available on some disk to receive the output. If any of these conditions fails, processing is deferred until the situation can be corrected.

As the simulation is presently implemented, the merge network is assigned whenever two lists are ready for merging, and merge processing is not interrupted before its completion. File S, however, cannot be processed unless the merge system is free. As a result, when the memory gets full, all files in core are combined into a single long intermediate result before File S is processed. During this period of consolidation, memory space can be released as unwanted data items are eliminated. If as a result of this process, the amount of free memory rises above the threshold value, new inputs can again be accepted from the disk. There is reason to believe that these policies leads to inefficiencies and that further algorithmic refinements are in order. See section 4.4.

3.5.5 Result Processing

All results are retained in core except those which involve File S and which therefore are left on disk. Results retained in core become

available for further processing; those on disk constitute a new "longest list".

In a practical retrieval system, the length of the file which results from a particular coordination procedure depends upon the operation being performed and upon the number of postings which are common to the two input lists. In order to simulate the effects of element duplication, an overlap factor, c_i, has been associated with each term, i. This factor reflects the extent to which term i indexes documents in common with other terms of interest. Using the notation of section 3.4, the length and overlap factor for the output file from the search "L1 OR L2" are given by

$$\ell_r = \ell_1 + (1-c_m)\ell_2, \qquad c_r = c_1 ,$$

where $\ell_1 \geq \ell_2$ and $c_m = \max(c_1, c_2)$.

Corresponding equations for AND and NOT processing are not used in the present study. If this rule is applied repeatedly in a search involving many terms, the length of the final result depends upon the order in which the terms are processed. Experimentally, this has not proved to be a serious problem: the result length from trial to trial has been found to deviate from the overall mean value by only a few percentage points.

3.5.6 Standard Parameters

Disk-related parameters used throughout this study are those shown in Table 2.2. In addition, standard values of 10% for the overlap factor and 10% for the memory threshold have been adopted. A merge is assumed to require $\ell_1 + \ell_2 + 1$ hardware cycles to complete.

3.6 Example

The unified operation of the procedures discussed in this chapter is best described by means of an example. Suppose a search request has been received for any document indexed by one or more of eleven specified terms. Table 3.1 shows the number of documents posted to each term and also the time interval after the start of the search during which each file will first be available. A standard merge system with 16 parallel data paths and a 6K word memory has been assumed. Table 3.2 lists the important events which occur during the progress of the search. Many of the essential time relationships are illustrated graphically in Figure 3.2. For each of the three rotations required to process this request, the figure shows the initial arrangement (in time) of data on the disk and the distribution of merge activity during the period. Element heights in Figure 3.2 are not significant but have been chosen merely to differentiate between adjacent or overlapping activities.

During the first rotation all but three and part of a fourth of eleven original lists are processed, and the merge network is occupied for 16.3 out of 25ms. The remainder of the process requires about 1-1/2 additional rotations and 23.8ms of additional merge time.

Term	Postings	Start Address (ms. past reference)	End Address (ms. past reference)
L1	1976	0.500	2.222
L2 (File S)	2384	5.014	7.084
L3	199	5.167	5.347
L4	292	6.875	7.139
L5	750	12.236	12.889
L6	1680	12.570	14.028
L7	1600	15.556	16.945
L8	220	17.222	17.417
L9	100	17.431	17.528
L10	1414	21.445	22.681
(L10A	1280	21.445	22.556)
(L10B	134	22.556	22.681)
L11	156	21.806	21.945

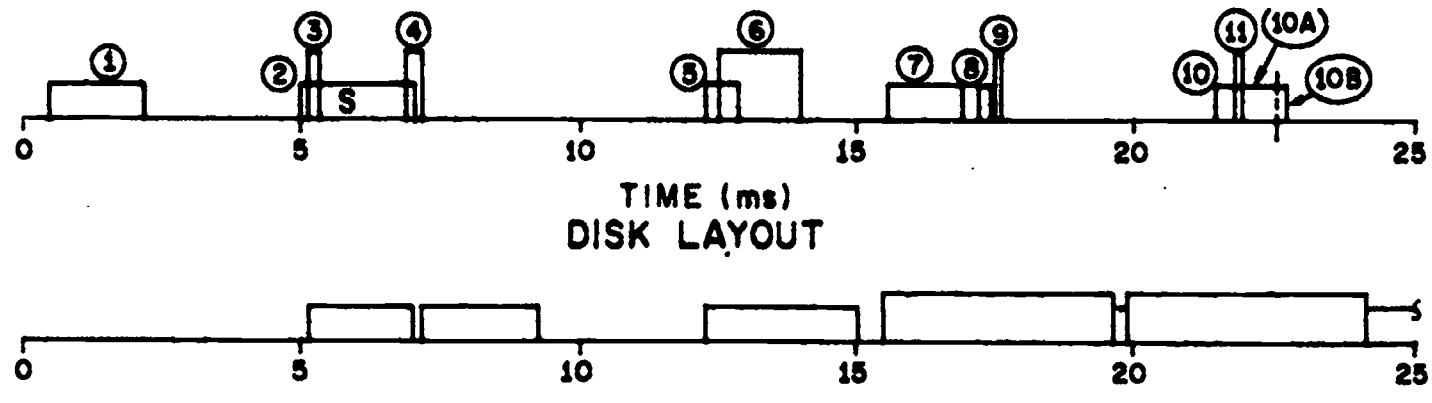
Table 3.1. Definition of Sample Search

Time (ms. past reference)	Event
0.000	Start of search
0.500	Read L1
5.014	Skip L2 (File S)
5.167	Read L3 and start merge (L1 and L3)
6.875	Read L4 and hold in core
7.084	End merge: result = T1 (2155 postings)
7.139	End read L4 and start merge (T1 and L4)
9.292	End merge: result = T2 (2417 postings)
12.236	Read L5 and start merge (T2 and L5)
12.570	Skip L6 (Read channel busy)
15.014	End merge: result = T3 (3092 postings)
15.556	Read L7 and start merge (T3 and L7)
17.222	Read L8 and hold in core
17.431	Read L9 and hold in core
19.653	End merge: result = T4 (4532 postings)
19.653	Start merge (L8 and L9)
19.959	End merge: result = T5 (310 postings)
19.959	Start merge (T4 and T5)
21.445	Split L10. Read L10A and hold in core
21.806	Skip L11 (Read channel busy)
22.556	End Read L10A
22.556	Skip L10B (Memory full)

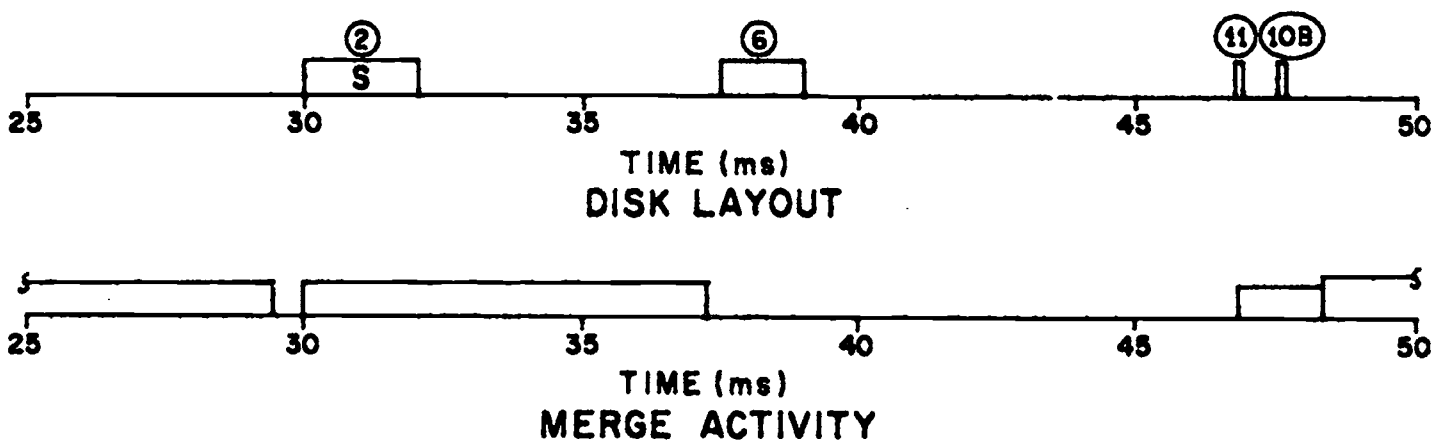
Table 3.2. Progress of Sample Search

Time (ms. past reference)	Event
24.195	End merge: result = T6 (4811 postings)
24.195	Start merge (T6 and L10A)
25.000	End of Rotation 1
29.500	End merge: result T7 (5963 postings)
30.014	Read L2 and start merge (T7 and L2)
37.278	End merge: result = *R1 (on disk) (8108 postings)
37.292	End write *R1
37.570	Read L6
46.806	Read L11 and start merge (L6 and L11)
47.556	Read L10B
48.417	End merge: result = T8 (1820 postings)
48.417	Start merge (T8 and L10B)
50.000	End of Rotation 2
50.139	End merge: result T9 (1940 postings)
55.052	Read *R1 and start merge (T9 and *R1)
63.792	End merge: result = *R2 (on disk) (9854 postings)
63.806	End write *R2. End of search.

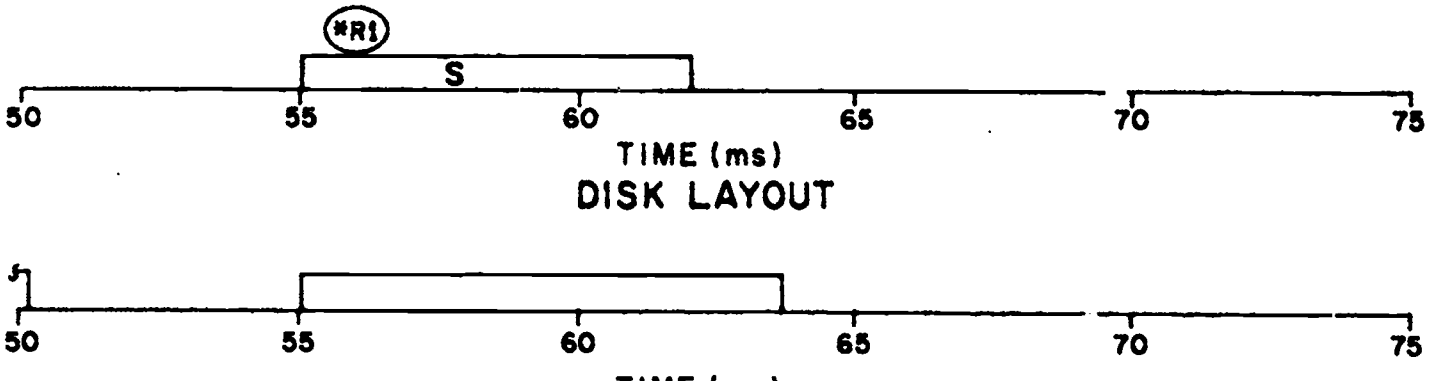
Table 3.2 (continued). Progress of Sample Search



(a) First Rotation



(b) Second Rotation



(c) Third Rotation

Figure 3.2. Processing Example: "OR" Eleven Terms

4. PERFORMANCE

4.1 Preliminaries

The goal of the performance studies reported in this chapter is to assess the capabilities of the new system operating in a realistic retrieval environment. To this end, sample searches have been selected from the full MEDLARS Master MESH as of November, 1972, [15] with the aid of [3] and other data obtained partly from the National Library of Medicine describing the MEDLARS and MEDLINE retrieval systems. Any errors in interpreting this information, of course, lie entirely with the author of this report.

While characteristics of the MEDLARS data base have been used to add realism to these tests, the simulated system differs in certain regards from both MEDLARS and MEDLINE and is not intended to be a direct representation of either one.

The Master MESH is a listing of the complete Medical Subject Headings (MESH) Index together with a tally of the documents indexed under each term. The MESH index language is a carefully controlled, hierarchically structured vocabulary of over 8500 terms employed by professional indexers to classify technical articles from 2200 journals. The data base used in this study contains over 1,000,000 citations dating from January 1964 to November 1972. Individual terms reference from one to nearly 500,000 documents. It is interesting to note that the data base for MEDLINE, the on-line version of this service, currently contains about 450,000 citations and is limited in coverage to approximately the most recent three years. This restriction is necessary in part because of the prohibitively long search times required to process the larger data base in an on-line, real

time environment and still provide adequately fast response for a large and growing number of users.

The search language employed in the MEDLARS system permits the retrieval of all documents indexed under one or more terms joined by the logical connectives "AND", "OR", and "AND NOT" in a standard way. Several techniques exist for modifying the basic search pattern, and one--the explosion--is of special interest here. The request EXPLODE (TERM) is a shortcut for searching simultaneously a general term and all of its subordinates in the hierarchy. It accomplishes the same objective as ORing all terms with the same classification number. More than one EXPLODE can be included in a search statement, and such requests can result in searches involving a very large number of terms and a large amount of processing time.

The primary example used in these studies is the moderately long search,

EXPLODE (CENTRAL NERVOUS SYSTEM),

which involves the coordination of 70 terms having a combined total of 67,527 document postings (see Table 4.1). A shorter search,

PARALYSIS OR PARAPLEGIA OR QUADRIPLEGIA,

(Table 4.2) is mentioned occasionally for comparison. Currently, short searches occur more frequently than longer ones although both are common. However, as data bases expand and user communities grow, demands on a retrieval system increase. This analysis is oriented toward the longer search because it provides a better description of the system's performance under heavy load.

Term	Documents Referenced
CENTRAL NERVOUS SYSTEM	2976
BRAIN	16277
BRAIN STEM	1680
MEDULLA OBLONGATA	1125
OLIVARY NUCLEUS	199
PONS	750
CEREBELLOPONTILE ANGLE	156
VESTIBULAR NUCLEI	292
RETICULAR FORMATION	1254
CEREBELLUM	2000
DIENCEPHALON	812
HYPOTHALAMUS	4931
HYPOTHALAMO-HYPOPHYSEAL SYSTEM	1440
MAMMILLARY BODIES	24
THALAMUS	1737
GENICULATE BODIES	845
THALAMIC NUCLEI	395
MESENCEPHALON	1471
CORPORA QUADRIGEMINA	403
INFERIOR COLLICULUS	93
OPTIC LOBE	226
SUPERIOR COLLICULUS	171
RED NUCLEUS	186
SUBSTANTIA NIGRA	294
TELENCEPHALON	434
CEREBRAL CORTEX	6877
CORPUS CALLOSUM	472
FRONTAL LOBE	818
GYRUS CINGULI	106
MOTOR CORTEX	313
OCCIPITAL LOBE	553
VISUAL CORTEX	1448
PARIETAL LOBE	398

Table 4.1. Data for Long Search [15]

Term	Documents Referenced
SOMATOSENSORY CORTEX	281
TEMPORAL LOBE	619
AUDITORY CORTEX	502
HIPPOCAMPUS	1515
AMYGDALOID BODY	526
LIMBIC SYSTEM	1077
CEREBRAL VENTRICLES	1655
CEREBRAL AQUEDUCT	35
CHOROID PLEXUS	363
CISTERNA MAGNA	172
EPENDYMA	338
MENINGES	344
ARACHNOID	243
SUBARACHNOID SPACE	305
DURA MATER	513
PIA MATER	145
NEURAL ANALYZERS	257
SPINAL CORD	3773
CAUDA EQUINA	195
EXTRAPYRAMIDAL TRACT	294
PYRAMIDAL TRACTS	376
SPINOTHALAMIC TRACTS	28
ANTERIOR HORN CELLS	170
AUDITORY PATHWAYS	102
CRANIAL FOSSA, POSTERIOR	234
TEGMENTUM MESENCEPHALI	48
VISUAL PATHWAYS	257
LISSAUER'S TRACT	4
NEURAL INTERCONNECTIONS	900
POSTERIOR COLUMNS	4
RESPIRATORY CENTER	211
CEREBELLAR CORTEX	588
CEREBELLAR NUCLEI	126
CORPUS STRIATUM	72

Table 4.1 (continued). Data for Long Search [15]

Term	Documents Referenced
SEPTAL NUCLEI	9
OLFACTORY BULB	75
OLFACTORY PATHWAYS	15
Total	67527
Average	965

Table 4.1 (continued). Data for Long Search [15]

PARALYSIS	2026
PARAPLEGIA	1033
QUADRIPLEGIA	374
Total	3433
Average	1144

Table 4.2. Data for Short Search [15]

The experimental procedure has been to generate a series of performance curves showing the average time required to perform a search under various conditions using various system configurations. Every point plotted explicitly represents the average of results from 30 trials where each trial involves a complete simulation of the search in question, beginning with the random assignment of a disk address to each data file and the random choice of an initial rotational position. Coordination proceeds according to the algorithms in the previous chapter under the control of a supervisory program which allocates system resources, performs the initial address assignments and collects performance data. Whenever multiple, independent searches are conducted simultaneously, each term in each search receives a separate disk address; and the coordination algorithms are applied to each search independently. Thus, if two of the long searches defined in Table 4.1 are processed in parallel, it is as if two users had requested searches having identical parameters but entirely different index terms. Independent searches compete for limited system resources such as memory space, the merge network and the I/O facilities.

The monitor program controls the progress of a simulation by maintaining a time-ordered queue listing all events of interest to the system. These include the access times for all files to be processed and the scheduled completion times for all I/O and merge procedures in progress. This routine can refuse any request for service if the required facilities are in use or if other conflicts arise. In this event, the postings file in question is considered again on the next rotation.

System configurations examined in these tests include 1, 16, 32, 64, 128, 256 and 512 parallel data paths and standard memory sizes of 4K, 8K, 16K, 32K, 40K, 50K, and 64K words (K=1024). Other memory sizes have

been used under special circumstances as noted in the text. Three configurations receive particular attention: in the remainder of this report the terms "large", "small" and "conventional" are used to describe systems having 256, 16 and 1 data path, respectively. As mentioned previously, a system with 256 parallel paths is the largest considered technologically feasible at the present time. A sixteen-path system performs well and should be relatively cheap and easy to build and maintain. The use of a one-path system to represent a conventional machine is somewhat arbitrary, but it is believed to be a conservative choice for the following reasons.

Consider a conventional movable head disk with a 25ms rotation time, a 60ms average track access time and a track capacity of 1800 words. Thirty-seven and one-half rotations, or approximately 0.94 seconds, are needed to transfer the data required for this problem. If the seventy files are located randomly on the disk, then, on the average, additional penalties of 0.88 seconds for latency and 4.2 seconds for head motion are required to access the data. Finally, on the basis of published execution times for a large general purpose computer[1] and a short segment of code written to perform the term coordination function on this machine, it is estimated that approximately six microseconds of processing time per data element are required to perform this task. At that rate, 2.3 seconds of CPU time are required to merge 64 lists of 1000 items each using an optimal 2-way merge. Adding these times yields a rough estimate of 8.32 seconds for a conventional machine to perform this search. No allowance is made for interruptions other than disk I/O, and a memory adequate to

---

[1] IBM 360/75 with four-way interleaved memory and IBM 2314 A-series disks.

hold all data is assumed. The corresponding figure from simulation of the one-path hardware merge system is 4.23 seconds. With a memory restricted to 4K words, the time for the one-path system is 9.47 seconds. Thus it is felt that the one-path simulation produces a time estimate which is comparable with but generally shorter than the processing time that would be required by a conventional computer with the same size memory.

4.2 Monoprogrammed Results

4.2.1 Basic Tests

Figure 4.1 contains the primary experimental result of this paper: a description of the time required to process a long search using hardware systems and data memories of various sizes under the standard conditions defined in Chapter 3. Presentation of the actual merge times associated with these trials is deferred until section 4.5 since those curves require some special interpretation. Since the longest file is never retained in core, a 64K word memory is sufficient to contain all the data which must be processed internally for the long sample search: no further increase in memory capacity can affect the results.

Figure 4.1 includes all the configurations discussed in the previous section except the conventional system, whose performance curve lies beyond the range of the graph. Data for that system appears under the headings "Conventional System" and "Mean" in Table 4.3, which also contains average values for the small and large parallel systems. The remaining columns in the table show the standard deviations for the various samples and the corresponding 95% confidence intervals, assuming that use of the Central Limit Theorem is justified. With a probability of 95%, every point

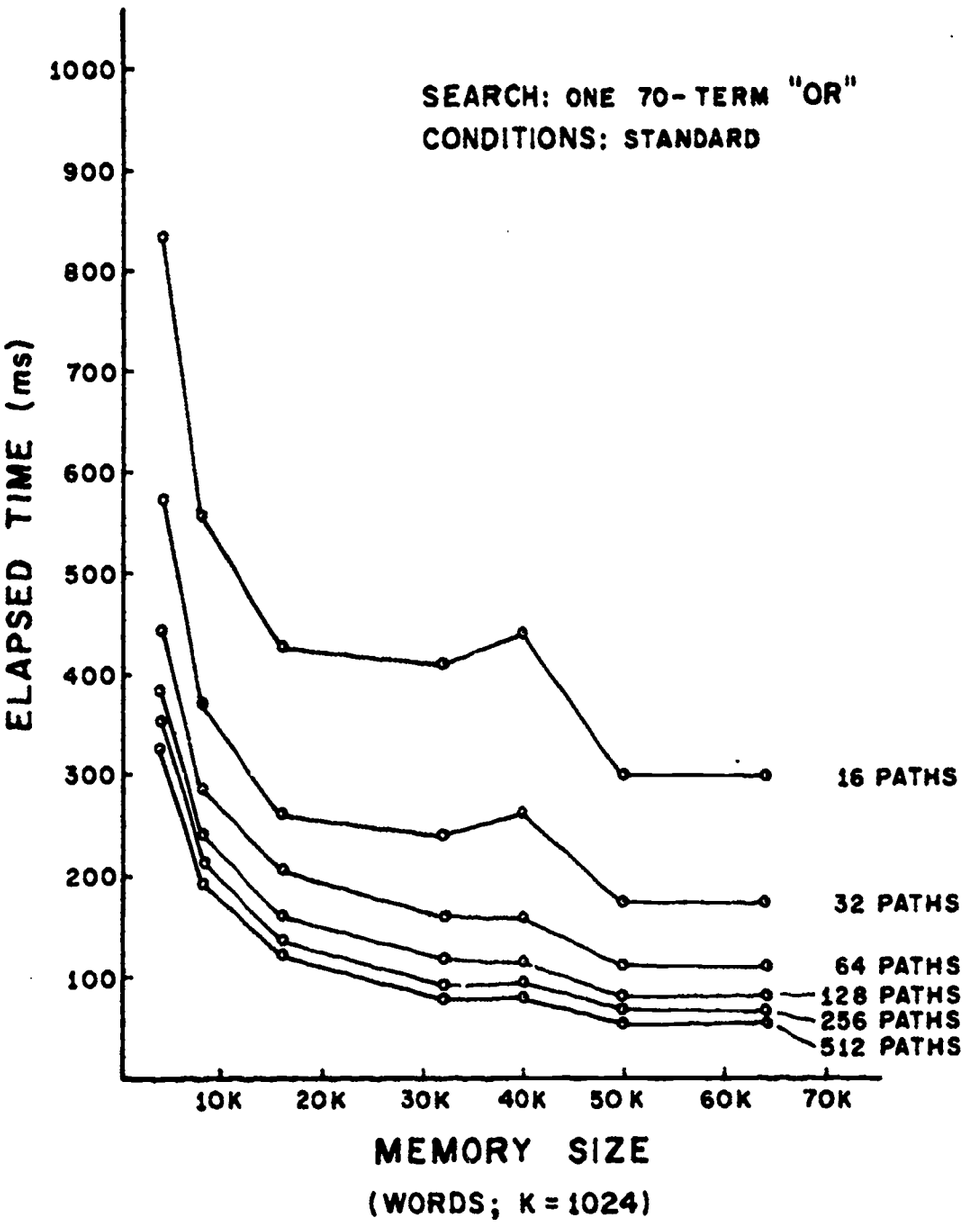


Figure 4.1. Basic Performance Analysis

Memory Size (Word)	Conventional System (One data path)			Small Parallel System (16 data paths)			Large Parallel System (256 data paths)		
	Mean	Std. Dev.	95% Confidence Interval for Mean	Mean	Std. Dev.	95% Confidence Interval for Mean	Mean	Std. Dev.	95% Confidence Interval for Mean
4K	9472.72ms	102.06ms	± 38.07ms	833.09ms	36.77ms	±13.72ms	357.93ms	28.40ms	±10.59ms
8K	6506.13	381.07	±142.14	559.22	37.83	±14.11	215.59	23.48	± 8.76
16K	5115.66	283.35	±105.69	426.91	36.97	±13.79	135.61	20.64	± 7.70
32K	5086.72	278.12	±103.74	410.55	38.20	±14.25	94.21	15.08	± 5.63
40K	5637.71	513.53	±191.55	442.58	40.48	±15.10	95.00	14.77	± 5.51
50K	4365.33	119.93	± 44.74	302.93	11.45	± 4.27	69.92	11.86	± 4.42
64K	4231.09	40.99	± 15.29	300.88	11.83	± 4.41	68.10	9.96	± 3.72

Table 4.3. Long Search Performance for Three Standard Systems

plotted for the large and small parallel systems lies within 15.1ms of the true average value. In most cases the interval is actually much smaller. For the conventional system, the confidence interval extends as far as ±192ms from the calculated average value, a deviation of less than 3.5% from the mean.

Ignore, for now, the local maximum which the performance curves exhibit around 40K words and consider the performance potential these results represent. Table 4.4 shows the speed-up which can be achieved by the various systems relative to the assumed conventional machine. For a small system, the speed-up is roughly a factor of 12 at all memory sizes; for a large system, the factor varies from 26.46 with a 4K memory to approximately 62 with a memory of 50K words or more. In absolute terms, the large system can coordinate 70 files containing a total of over 67,000 postings in an average time equivalent to about 2-1/2 disk rotations.

In this test, the large system outperforms the small one by a factor which ranges approximately from 2.5 to 4.5--a small improvement considering the additional cost, complexity and bandwidth involved. The reason is that this search does not represent a very heavy load for these machines, especially the larger one. A greater separation can be seen in some of the data base size experiments to be presented in the next section.

For comparison, Table 4.5 presents results achieved in processing the short, three-term, sample search. All figures in the table represent system configurations having sufficient memory to hold all data files. In processing a single search of this magnitude, the parallel systems outperform the conventional system by a factor of only about 3, and there is very litt › difference between the two parallel systems. When 10 independent

Memory Size (words)	16-Path ("Small System")	32-Path	64-Path	128-Path	256-Path ("Large System")	512-Path
4K	11.37	16.49	21.34	24.56	26.46	27.96
8K	11.63	17.46	22.60	27.04	30.18	33.91
16K	11.98	19.50	24.71	31.57	37.72	38.06
32K	12.39	21.14	30.59	42.19	53.99	63.81
40K	12.74	21.54	35.30	48.68	59.34	69.58
50K	14.41	24.92	37.76	53.67	62.43	76.76
64K	14.06	24.18	37.37	49.33	62.13	70.80

Table 4.4. Speed Improvement Factors Relative to Conventional Processor

Search	Conventional System	Small Parallel System	Large Parallel System
One Short Sample	102.92ms	34.19ms	30.42ms
Ten Simultaneous Short Samples	807.38	125.44	55.49

Table 4.5. Elapsed Time for Short Sample Search

short searches are processed simultaneously, performance differentials became more apparent; and improvement factors over the conventional machine range from 6.4 for the small system to 14.6 for the large one.

4.2.2 Data Base Expansion

One major concern in the design of any information retrieval system is its potential for growth: how rapidly will performance degrade as the data base expands? To answer this question, the same 70 term sample search was used, but the lengths of all the postings files were multiplied by factors of 1/4, 2 and 4; and new curves were generated for these modified data bases. Results are shown in Figures 4.2(a) and 4.2(b) for the small and large systems, respectively. Only one point (30 trials), which does not appear in the figures, was obtained for the conventional system because of the prohibitively high cost of simulating this configuration. Its average processing time is 16.81 sec. for the 4X expansion with a 220K word memory. This is slower than the corresponding small parallel system by a factor of 15.4 and slower than the large system by a factor of 160.7.

For reasons to be discussed in section 4.2.3, it is considered valid to compare these systems at the points where minima occur in the performance curves: 50K, 100K and >200K memory sizes. Figure 4.3 shows the average elapsed time for the search as a function of data base size and memory size for two parallel configurations. In both cases, the large-memory curves are very nearly linear while the 50K curves show an increasing slope with increasing data base size. This effect, which reflects degraded performance under heavy load, is much more pronounced for the small system than the large one. Considering the best performance available from both systems, a four-fold increase in the data base size (from X to 4X) increases

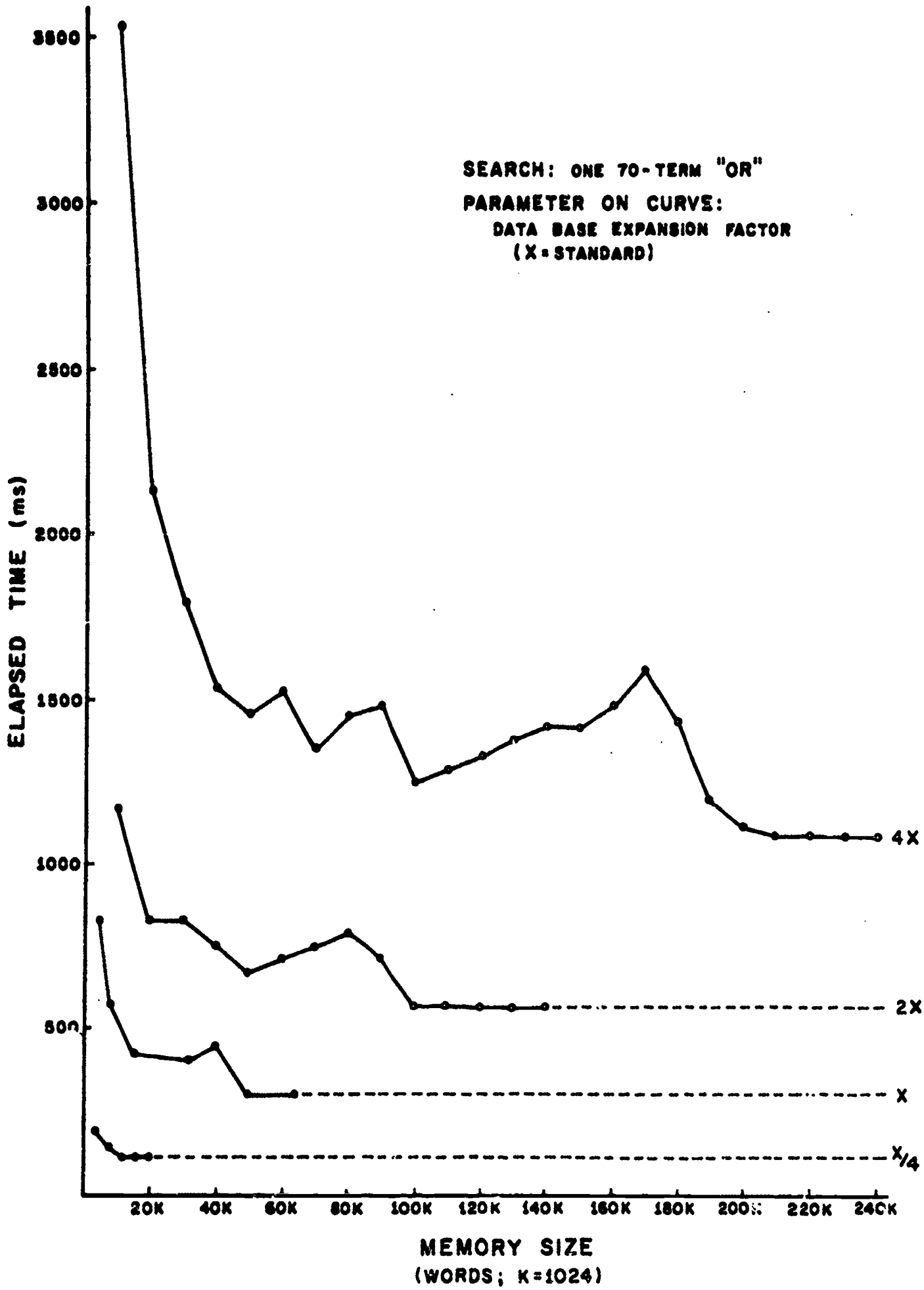
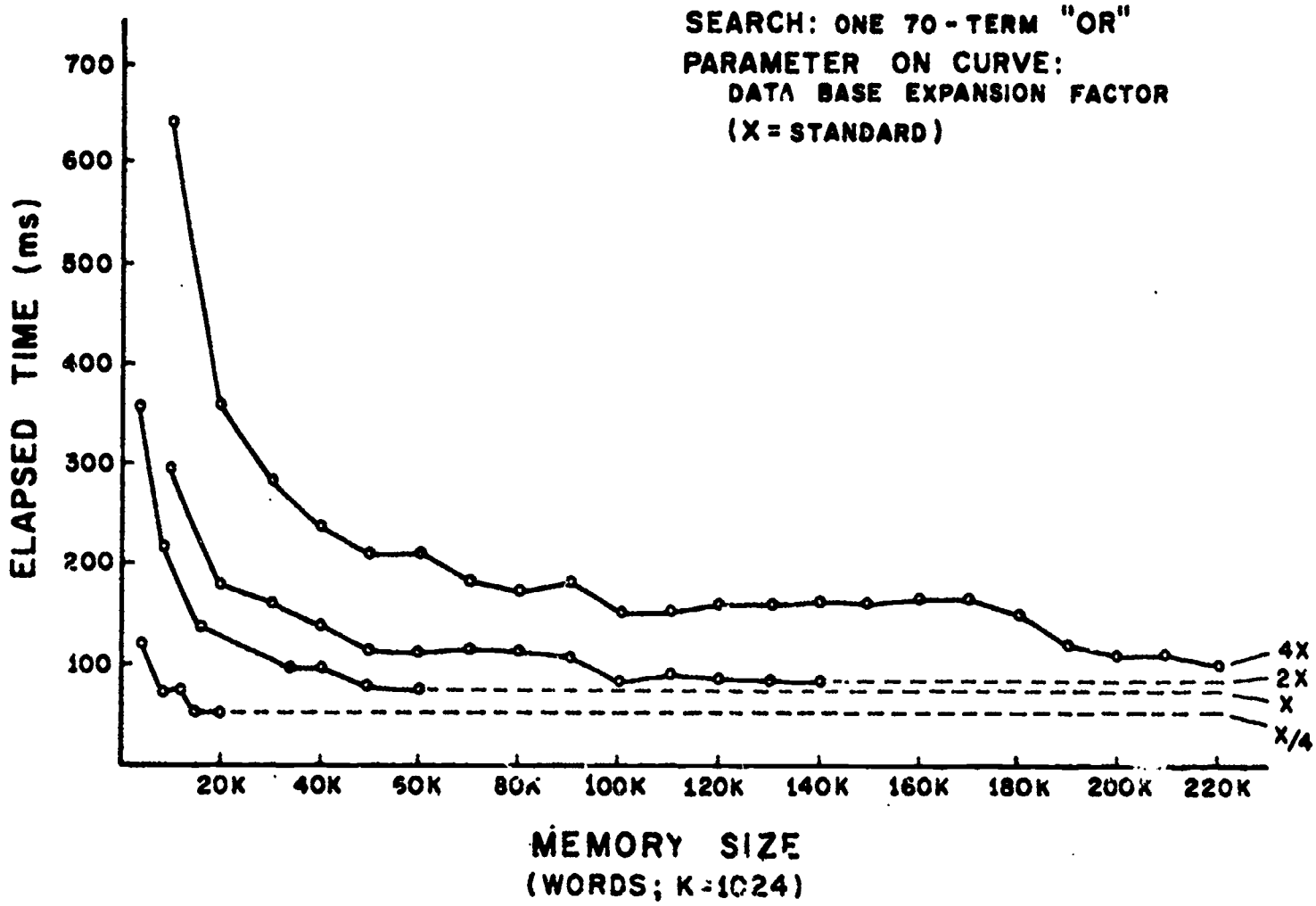


Figure 4.2. Effects of Data Base Expansion



(b) For Large System

Figure 4.2 (continued). Effects of Data Base Expansion

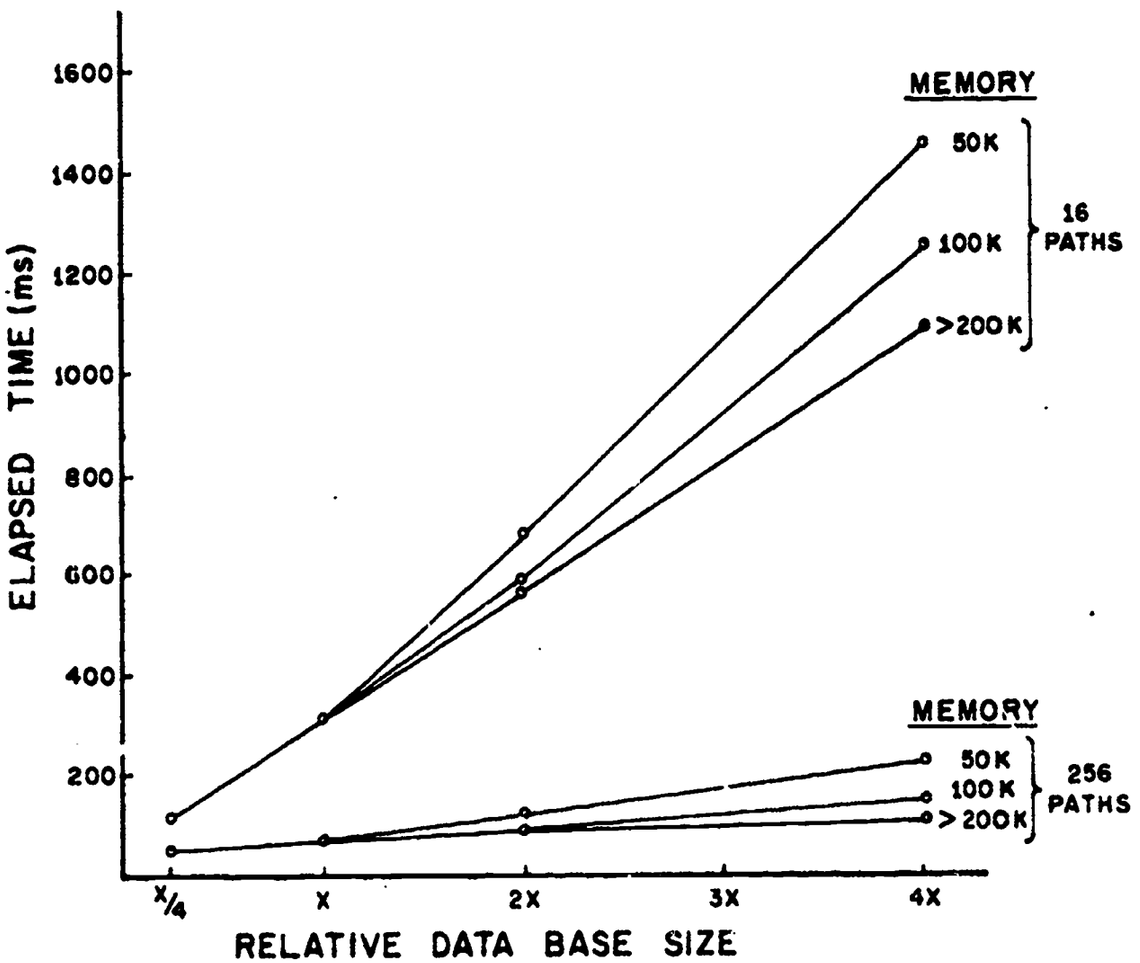


Figure 4.3. Comparison of Small and Large Systems with Expanded Data Bases

the response time for the small system by a factor of 3.6 (to about 1.0 second) and that of the large system by a factor of only 1.3 (to about 0.1 second). Evidently there is room in the large system for considerably more than a four-fold expansion in the assumed data base before response times on the order of a few seconds will be encountered.

4.2.3 Discussion of Performance Curves

All the performance curves presented thus far have exhibited certain common characteristics. For small memories, processing time decreases rapidly with increasing memory size up to a certain point. For large memories--large enough to hold all the files to be coordinated except File S--processing time reaches a constant minimum value. Between these two extremes a peculiar but very consistent system of oscillations occurs. These oscillations are a direct result of the necessity to alternately fill the memory with new data and then combine the resulting intermediate list with the output file on disk.

Consider again the 4X data base performance curves presented in Figures 4.2(a) and 4.2(b), shown together for convenience in Figure 4.4. The curve for the small system contains sharply defined minima at 50, 70, 100 and 210K, with distinct peaks at 60, 90 and 170K words. The curve for the large system contains a series of "plateaus" extending from 40-60K, 70-90K, 100-170K and upwards from 200K. Here the divisions are not as well defined as on the other curve because the peak-to-peak variation is much smaller. Nevertheless, the curve is clearly divided into several regions, and the region boundaries correspond closely to the minima on the first curve. Each of these regions corresponds to a different number of repetitions of the memory filling and clearing cycle.

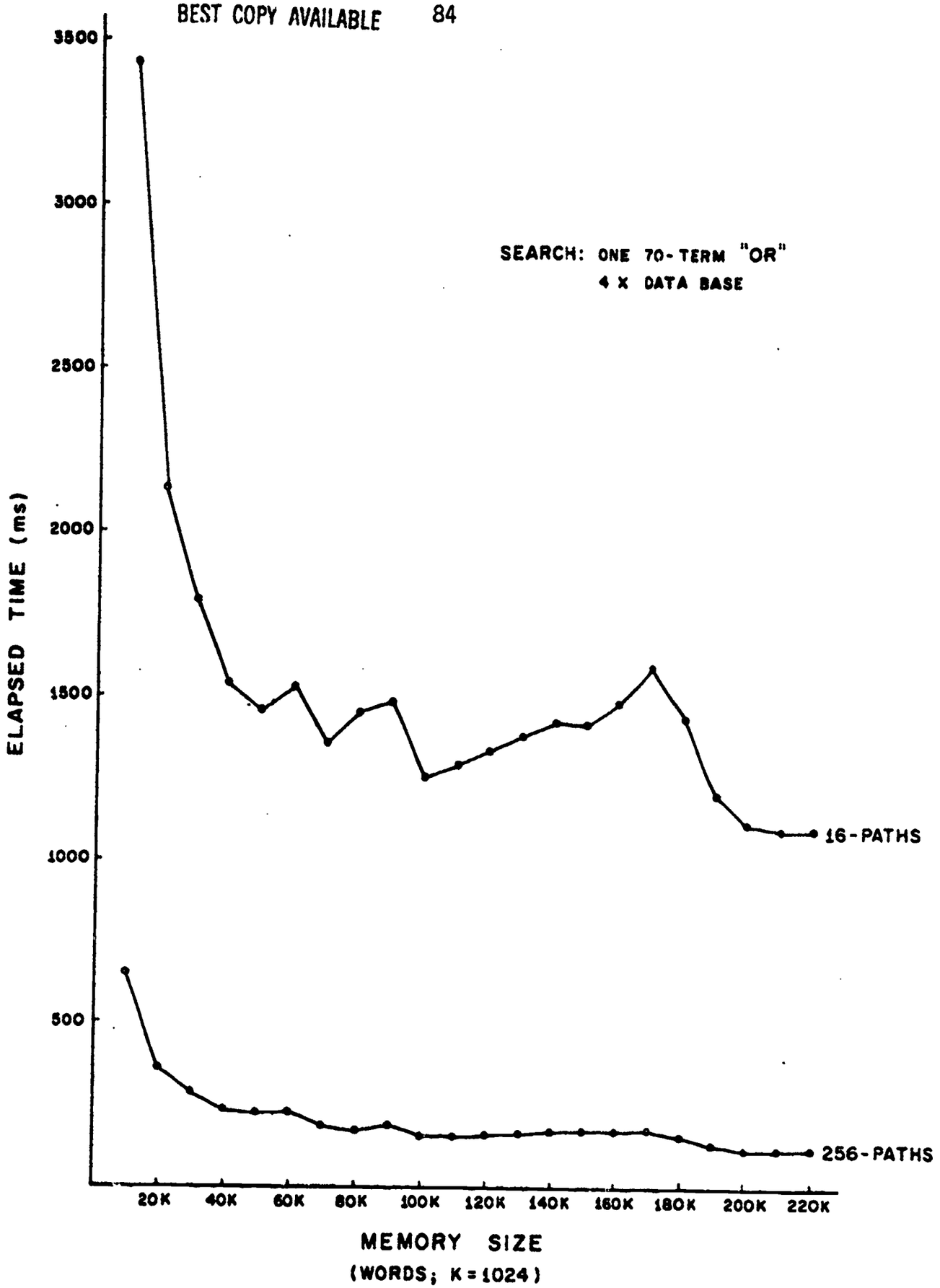


Figure 4.4. Comparison of Large and Small System Performance with 4X Data Base

Under the list selection algorithms defined previously, there exists some critical memory size, M_1, above which it is always possible to perform this search by filling the memory and processing the longest list only once. Similarly, there exist other values M_2, M_3, ..., above which the longest list need be processed no more than twice, three times, etc. These critical values are located approximately at the minima on the performance curve. Similarly, there exists a second set of critical memory sizes N_1, N_2, ..., N_j, etc., below which it is never possible to complete the search by processing the longest list 1, 2, ..., j times. These points correspond to the maxima on the performance curves.

Processing the longest list is a significant event in this system because it requires an unusually long merge (involving all the data in memory and all the postings on the current longest list), and it also entails a disk latency penalty of one-half rotation on the average. During all this time, no other list collection or processing can proceed.

From the curves in Figure 4.4, M_1 for the sample search lies around 210K. If so, one would expect to find M_2 near $M_1/2$ = 105K, M_3 at $M_1/3$ = 70K, M_4 near $M_1/4$ = 52.5K, etc. In fact, these are the observed locations of other minima on the curves.[2] As the memory decreases further in size, the critical points lie closer and closer together, the peaks tend to become smaller, and the curves rise very steeply.

---

[2]No data exists at 105K; minimum occurs at 100K.

Other performance curves behave in a similar fashion. Thus, in Figure 4.1, all curves exhibit a local maximum at 40K (except the 64 and 128 path systems, which never quite reverse their direction), and all reach their final minimum values at 50K. In Figures 4.2(a) and 4.2(b), only the 4X curves have critical points (minima) near 70K; the 2X and 4X curves both have critical points at 100K and 50K; and the X curve has critical points near 50K and 30K. The 2X curve may also contain a minimum between 10K and 30K, but the sampling interval is too large to show this clearly. These results are all consistent with the present theory.

Between points N_i and M_i there is a transition region in which it is sometimes necessary to process the longest list i times and sometimes i+1. In this area, performance improves rapidly with increasing memory size. Between the points M_i and N_{i-1} there is a larger region where the long list cycle is always repeated i times, but where larger memories may be less effective than smaller ones. This results from the interaction of several phenomena, most of which cannot be observed consistently to favor one memory or another. On the average, however, their combined effect is to discriminate against larger memories.

First, it takes longer to fill a large memory initially than a smaller one. Then, too, it takes longer with a large memory to process the last few sublists in core because these lists tend to be longer than they would be in a smaller memory. If, as the algorithms now stand, in the course of this final processing, the amount of free core rises above a certain level, a few new, relatively short lists may be read. They have to be incorporated with the long lists which are already there, a process which again favors a smaller memory. This threshold crossing can occur

several times, and it yields very little of value in exchange for the processing time it requires.[3] Processing the long list itself often takes longer with a large memory than with a smaller one because the processing time is proportional to the total number of data items which enter into the merge. Finally, during the last cycle of activity, the system with a larger memory tends to finish faster than one with a smaller memory because less data remains to be processed. This advantage of the large-memory configuration, however, is not sufficient to compensate for its several disadvantages.

This analysis of the performance curve may be useful in planning memory allocation procedures for processing multiple simultaneous searches. The following procedure is proposed without verification. Determine the combined total length of all files in the search except the longest one, and regard this number as an approximation to M_1. Then allocate a region size equal to $M_i = M_1/i$ (i=1,2,...), where i is determined by other factors such as the number of searches to be multiplexed, the desired response time, etc.

4.2.4 Other Parameters

Three other factors considered in this study which might influence system performance have all been found to be of minor significance. The effects of varying the overlap between postings files and of changing the

---

[3] A new algorithm which suppresses this activity has been tested with promising results. See section 4.4.

total processing time for the coordination of two files are considered in this section. Changes in the memory threshold are examined under Algorithmic Studies in section 4.4. In some cases these discussions are limited to the small parallel system when experimental results have shown that the effect of a given perturbation is more pronounced in the small system than in the large one.

4.2.4.1 Overlap

Overlap is a measure of the extent to which different terms index the same documents. In general, increasing the overlap factor decreases the effective size of the data base. This is clearly shown in Figure 4.5, which presents long-search performance curves for overlap factors of 0, 10% and 20%. For both large and small memories, processing time varies inversely with the overlap factor; between these extremes, the critical points tend to move toward smaller memories as the overlap factor increases.

4.2.4.2 Buffering Delay

It was shown in Chapter 2 that certain memory constraints can be relaxed if buffers are installed at the input and output of the special purpose hardware. The proposed change would increase the processing time for two lists from $\ell_1 + \ell_2 + 1$ to $\ell_1 + \ell_2 + 3$ cycles, where lists 1 and 2 contain ℓ_1 and ℓ_2 n-word sublists, respectively. Table 4.6 presents performance data for both the small and large systems with processing times of $\ell_1 + \ell_2 + 1$ cycles and a conservative $\ell_1 + \ell_2 + 5$ cycles. For this small variation, neither system is consistently faster, and the average

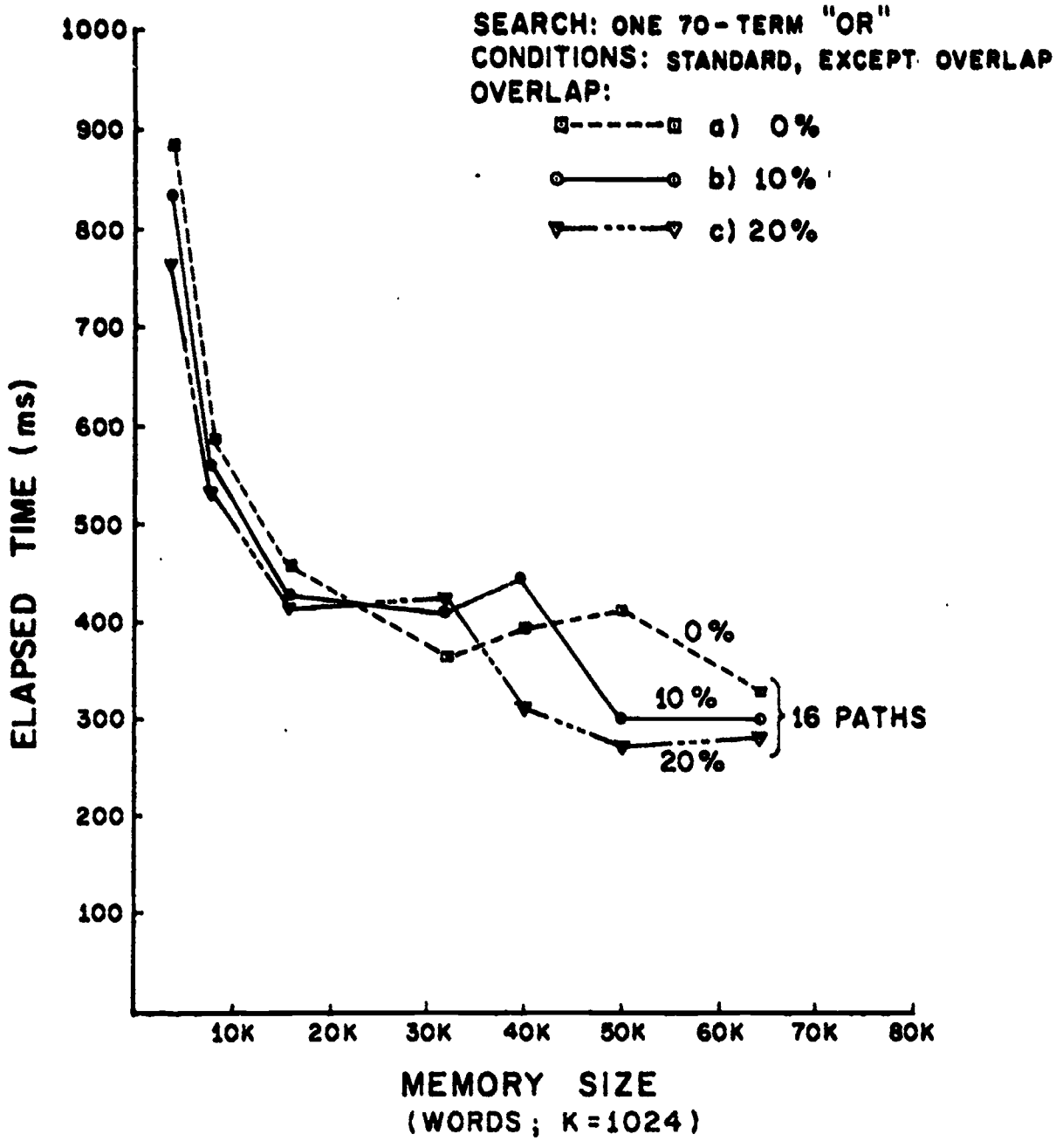


Figure 4.5. Overlap Factor Variations

Memory Size (words)	"Small" Parallel System — Processing Time			"Large" Parallel System — Processing Time		
	$\ell_1+\ell_2+1$ cycles	$\ell_1+\ell_2+5$ cycles	% Change	$\ell_1+\ell_2+1$ cycles	$\ell_1+\ell_2+5$ cycles	% Change
4K	833.09ms	846.38ms	+1.6 %	357.93ms	353.76ms	-1.2 %
8K	559.22	564.99	+1.0	215.59	212.79	-1.3
16K	426.91	415.29	-2.7	135.61	135.08	-0.39
32K	410.55	412.88	+0.57	94.21	95.89	+1.8
40K	442.58	444.58	+0.45	95.00	93.22	-1.9
50K	302.93	311.23	+2.7	69.92	70.28	+0.51
64K	300.88	301.82	+0.31	68.10	71.75	+5.4

Table 4.6. Variation of Processing Cycle Length

processing times differ by less than one-half rotation in every case except one.

4.3 Multiprogrammed Results

It is only possible at the present time to give an indication of the new system's capabilities for handling multiple simultaneous searches. In this situation the number of parameters and combinations of parameters which might be considered increases tremendously over the monoprogrammed case, and so does the cost of simulation. This discuss .n, therefore, may be regarded only as a point of departure.

Two important parameters--average response time as seen by the user and average elapsed time per search as seen by the system--and a simple model for performance evaluation will be considered. Average response time is defined to be the average time required to process a particular search request, and is closely related to user satisfaction. Average search time is simply the total time required to process a batch of n searches, divided by n, without regard for the completion times of individual searches. It is a measure of system throughput. To see the difference between average response and average search time, consider two searches processed concurrently beginning at time t=0 and ending at t=4 units and t=5 units, respectively. The average response time is 4.5 units, but the average search time is only 2.5 units. In applying these definitions in the present analysis, no allowance is made for time spent in preliminary processing (parsing and index file access) or for waiting time spent in various queues, which may be considerable. The object here is to examine those time requirements that are related directly to the use of the hardware coordination system.

Figure 4.6 presents a pair of hypothetical curves for response and search time as functions of the number of searches processed. Also shown are two broker lines: one is the constant value $t=t_1$, the average time required to process a single search by itself, and the other is the function $t_r = \frac{t_1}{2}(n+1)$, which represents the average response time that would be experienced by n users if their requests were submitted simultaneously and processed individually in sequence, each with processing time t_1. As long as the average search time for a group of searches is less than t_1, throughput can be increased by multiprogramming. The best performance, system-wise, occurs at the minimum on the search time curve. As long as the multiprogrammed response curve lies below the line t_r, users will experience improved performance over a monoprogrammed system with a comparable work load.

The somewhat limited test results which are available are shown in Figure 4.7. All tests were conducted with a 16-path system and approximately 64K words of memory (exceptions are noted below). Essentially similar results have been obtained for a 256-path system except that the times are shorter for tests involving the long search.

Part (a) of Figure 4.7 shows average search and response times for batches of from one to ten short searches. Over this range of work loads, the average response time rises only from 34.19ms for a single search to 71.77ms (three disk rotations) for a group of ten. This is well below the monoprogrammed reference curve. At the same time, the average search time drops from 34.19ms to 12.54ms and is still falling at the ten-search level, indicating that the most efficient operational load has not

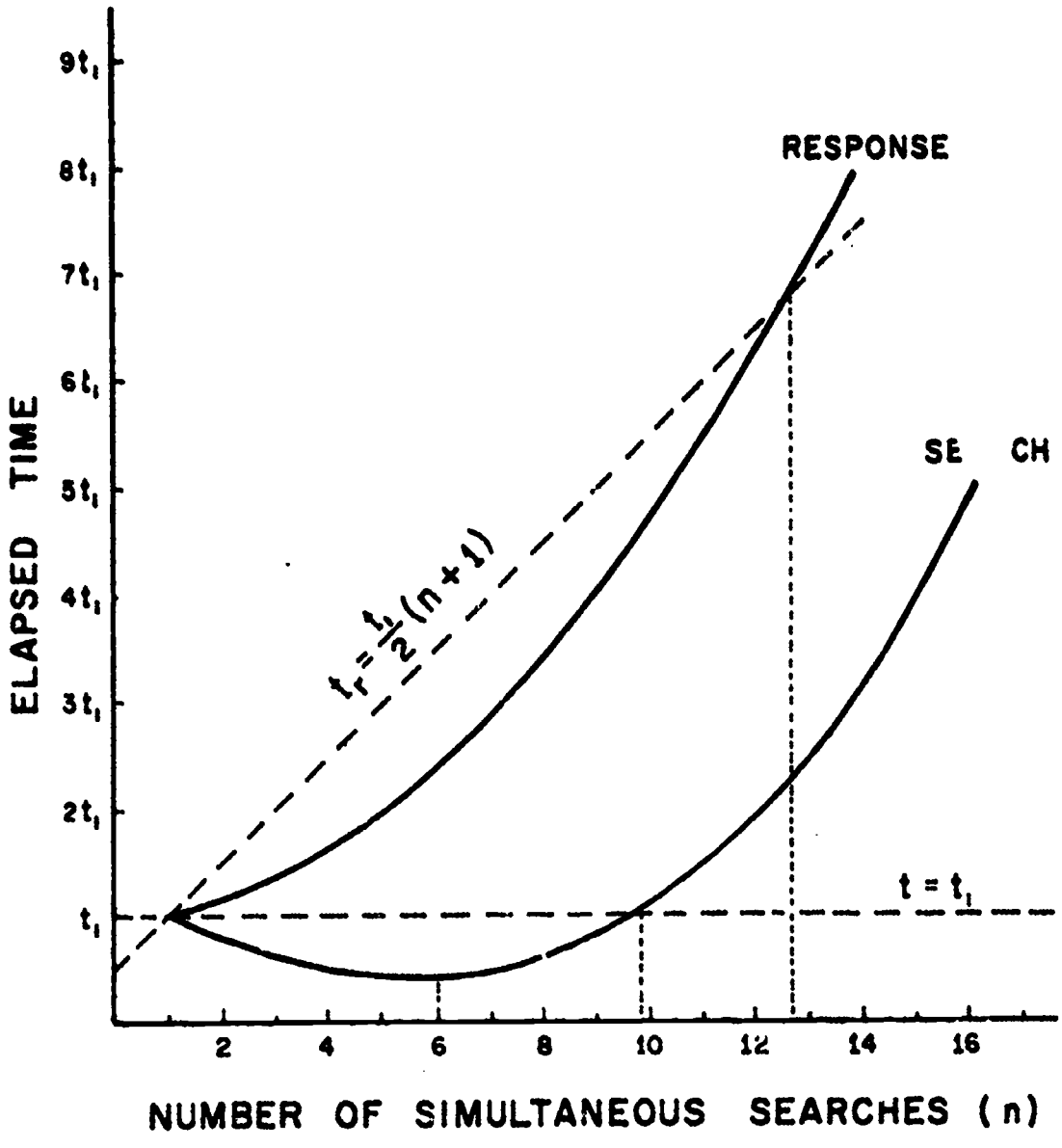
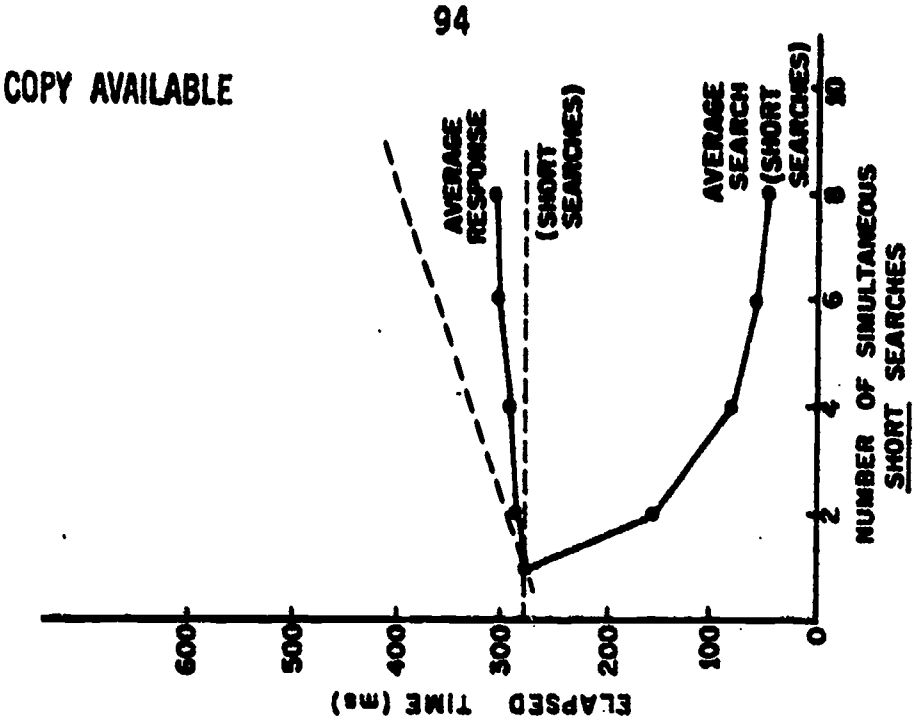
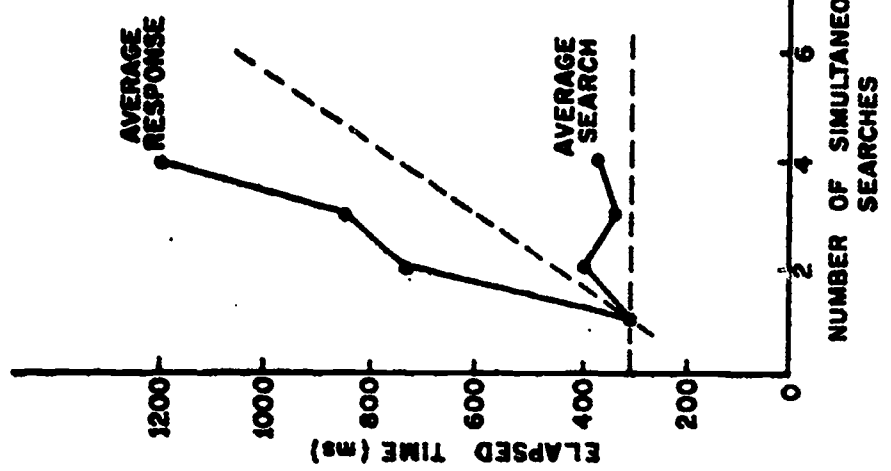


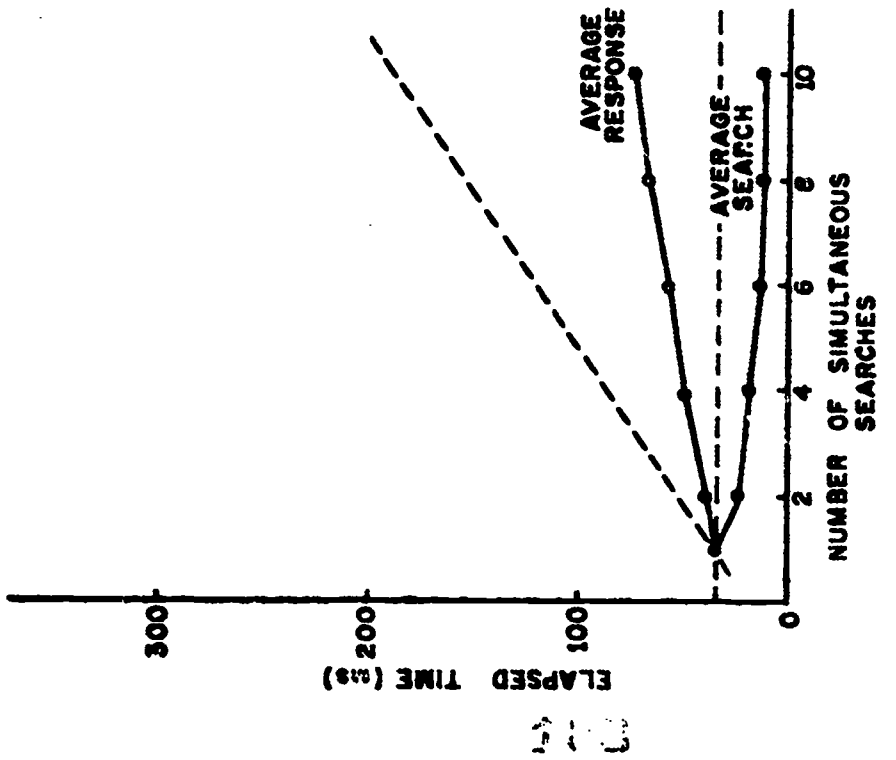
Figure 4.6. Average Search and Response Time Presentation



(c) Parallel Short Searches with One Long Search



(b) Parallel Long Searches



(a) Parallel Short Searches

Figure 4.7. Multiprogrammed Results

yet been reached.

Part (b) presents the corresponding results for loads of from one to four parallel long searches. In this case multiprogramming appears to be detrimental since both the average search and average response time curves lie above the corresponding curves for a monoprogrammed system.

For tests reported in Figure 4.7(b), each search was assigned a fixed memory partition before the start of processing. This memory allocation procedure was found to be superior to the system used for all other tests in which the several searches compete for available core on a dynamic basis. Partitions used are for one search, 52K; two searches, 40K; three searches, 24K; and four searches, 16K. Attempts to employ critically-sized regions as discussed in section 4.2.3 produced inconclusive results in which performance was sometimes improved by the use of critical region sizes and sometimes not.

Part (c) of the figure presents results for a mixed job load containing one long search and a number of short ones. This is likely to be a very common situation for an operational system. In this case, the monoprogrammed average response curve is assumed to have the same slope as in part (a) for parallel short searches alone. The important point is that the response and search time curves behave in much the same way in the presence of a long search as they do without one. Specifically, average response time increases by about thirty ms as the number of short searches increases from one to eight; and the average time per short search drops consistently throughout this range, indicating that more short searches could be included without serious adverse effects.

Further testing is required to develop a clear picture of system

behavior when processing multiple searches in parallel. Results presented in this section do, however, indicate the level of performance which can be achieved.

4.4 Algorithmic Development

Up to the present time, algorithmic development and refinement have been performed on an empirical basis. No claim is made to an optimal solution; however, certain experimental observations are worthy of note. In general, procedures have been avoided which would be difficult to implement or time-consuming to execute in an operational system.

The general problem is to merge N ordered lists of various lengths located initially at random positions on one or more disk drives. The available memory may or may not be adequate to contain all the data elements to be processed, but in general it is not.

In the initial experiments, all lists were processed strictly in their order of occurrence. When the memory became full its contents were combined with the next available list and the result was left on disk. For large problems, this procedure eventually resulted in a need to process a number of intermediate results, each too large to fit in core, located randomly on the disk. This proved to be a time-consuming job, and better performance was achieved when a single list (the longest) was reserved from the start for collecting intermediate results.[4] Curve A in Figure 4.8 was produced using this procedure.

---

[4]Cases have been observed in which performance is improved if the collector list is not chosen until it is needed, i.e., if the longest file is treated like any other until after the memory has been filled once, and the longest remaining list is chosen as a collector. However, this procedure has no clear-cut advantage over reserving the longest list from the start; and the latter is easier to implement.

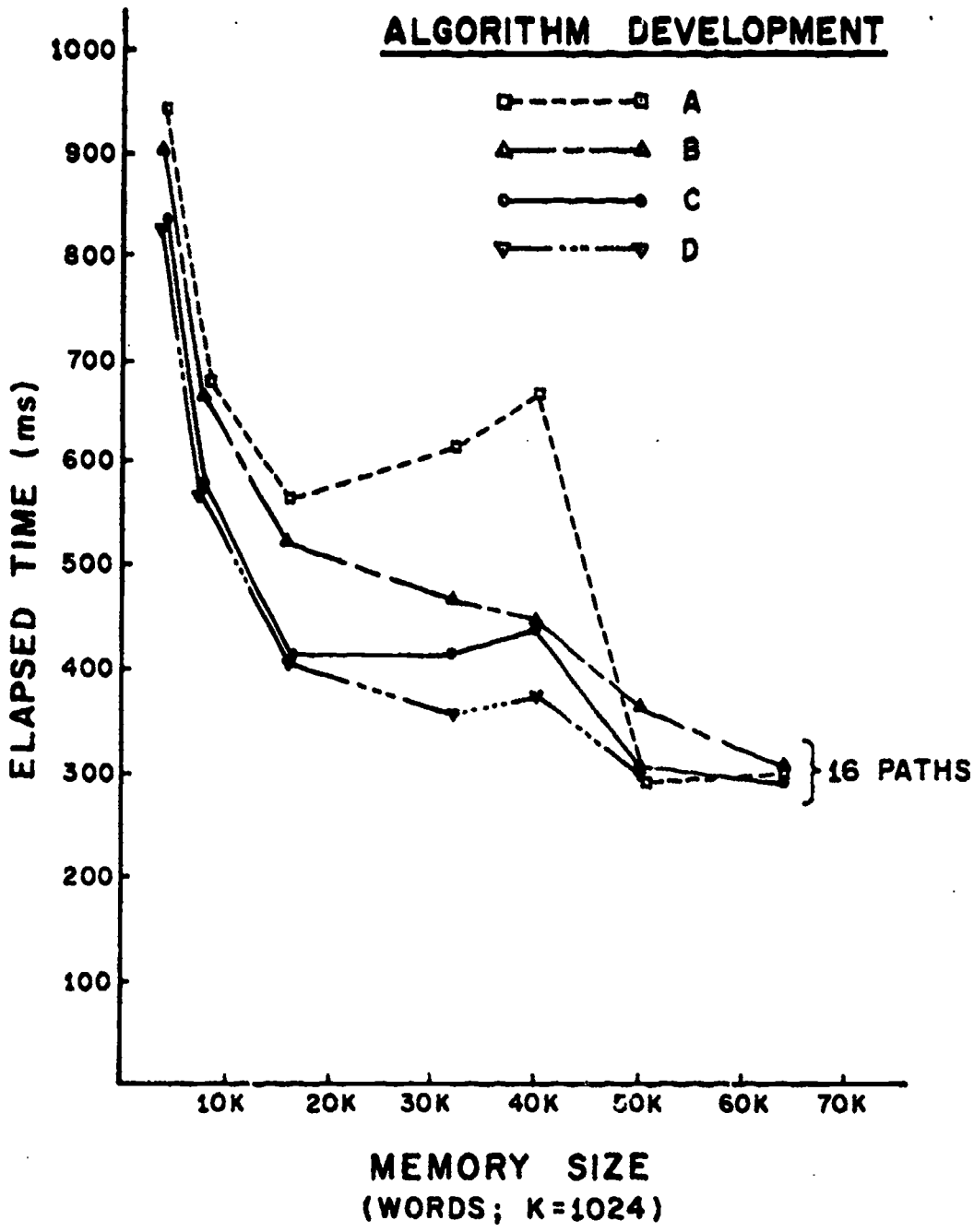


Figure 4.8. Algorithm Development

Curve A exhibits a large "hump" in the range of memory sizes between 16K and 50K words. In this operating region, it was found to be a very common occurrence for the memory to fill within a few blocks of its capacity and for the system to split a list in order to fill that small space. A long merge would follow and, because of the assumed overlap between the two lists, a few blocks of core would again become available at its completion. The whole process would then be repeated. Eventually the result would grow long enough to fill the memory completely, but meanwhile a great many opportunities for more useful work would be missed. To correct this problem, list-splitting was suppressed whenever the amount of free memory dropped below some threshold level, t. Curve B is the result for t=10%.

System performance was found to be fairly insensitive to the exact value of t above some low level. Curves for t=5, 10 and 20% all lie close together at most points tested. As one might expect, critical points show a tendency to shift towards large memories as t increases, indicating a reduction in the effective memory size of any given configuration.

In the next refinement, two additional activities were keyed to the level of memory usage. First, all reading of new data was suppressed when free core dropped below the threshold. More important, collector list-processing was permitted only when less than t% of the total memory was free. In this way a number of unnecessary, long merge procedures were eliminated and earlier access was permitted to short lists located at the same rotational position as the collector list. These improvements yielded Curve C of Figure 4.8, and this procedure has been used to generate all test results discussed elsewhere in this report.

Examination of the merge trees, or patterns of list combinations produced by Algorithm C reveals that, as the memory fills, the level of memory occupancy oscillates back and forth about the threshold, with the result that inefficient merges involving one very long and one very short list still occur frequently enough to degrade performance.

Recently (too recently for extensive use in this report), tests have been conducted using a new algorithm in which, essentially, no further reading is permitted after the memory fills beyond the threshold level.[5] Instead, all pending work on files in core is completed and the sink list is processed at the earliest opportunity. This algorithm was used to generate Curve D in the figure. Curve D lies below all other curves at all points tested and yields a particular improvement in the middle range of memory sizes. It does not, however, completely eliminate the local maximum at 40K which results from inefficient handling of long lists in memory sizes between the critical values M_2 and N_1, defined in section 4.2.3.

In view of the successful tests with Algorithm D, it now appears that the threshold system might be abandoned entirely in favor of a procedure which fills the available memory completely and then allows the work in progress to be completed and the memory emptied before accepting any new inputs. Other possibilities are also being considered.

---

[5]The precise procedure tested is that reading is suppressed after either a) the memory fills completely, or b) an opportunity to split a list is rejected because less than 10% of the memory is free.

4.5 Merge Activity

Records of merge and coordination hardware utilization have proved less interesting than expected, although they do support some of the analysis presented earlier. Figure 4.9 shows merge time as a function of memory size for the collection of basic test runs presented in Figure 4.1.

For small systems (1 (not shown), 16 and 32 paths), the merge time curves have nearly the same shape as those for total elapsed time. In particular, they exhibit a local maximum near 40K which, as discussed previously, results fr. inefficient merge scheduling as the memory becomes nearly full. When the memory becomes large enough to hold all files of interest, this inefficiency disappears, and the merge time drops to it overall minimum value.

For larger systems (64 or more parallel data paths) the phenomenon above becomes much less pronounced in the total elapsed time curves, and it disappears altogether from the merge time records. In fact, for systems with 128 and 256 paths, the merge time shows a definite increase around 50K (where total elapsed time decreases suddenly). Merge time for the largest system tested (512 paths) increases almost linearly with memory size above 8K while the total elapsed time decreases steadily. Thus, the efficiency of network utilization drops steadily as the overall performance of the system improves.

The explanation for this may be found by considering the nature of these large systems and examining the detailed progress of a search. In these configurations a great many data items are transmitted simultaneously from disk with the result that any given postings file occupies a much smaller angular region than would otherwise be the case, and the

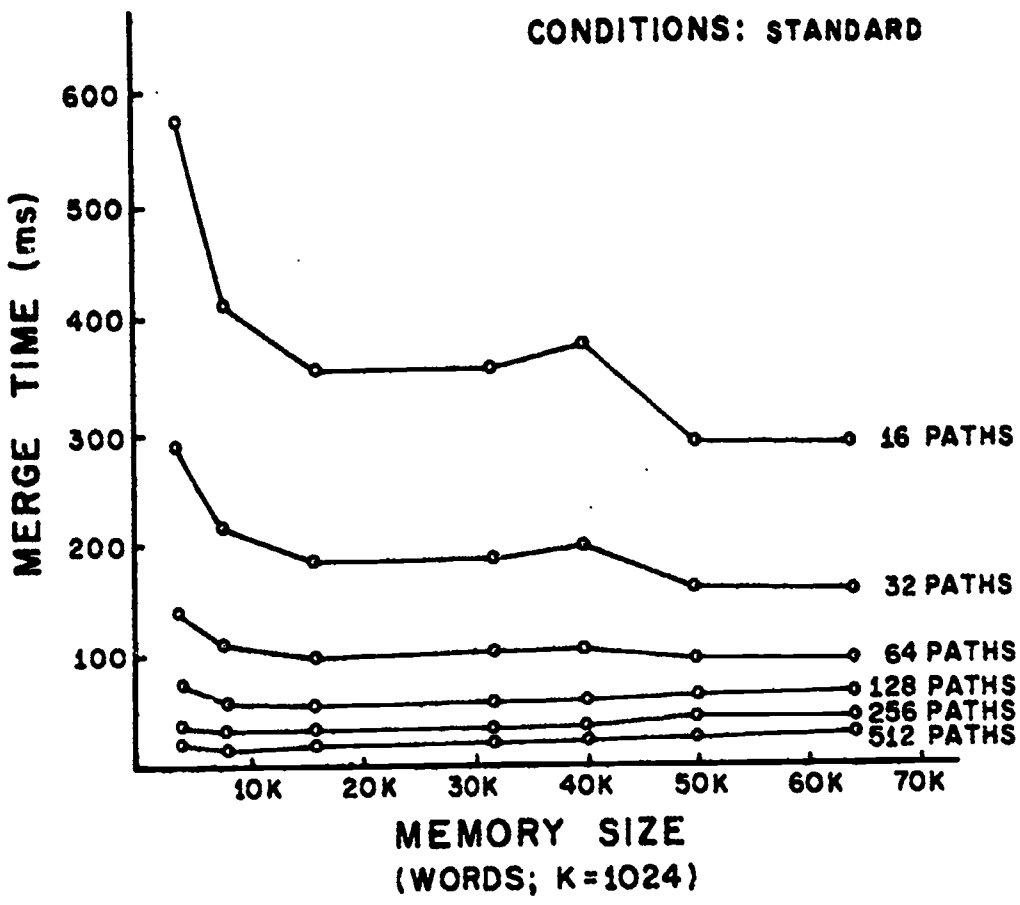


Figure 4.9. Merge and Coordination Hardware Utilization

disk appears to be less densely populated. Furthermore, because the hardware coordination system also processes more data on each cycle, a given merge is completed more rapidly. These two factors combine to prevent the accumulation of unprocessed lists in core so that instead of forming small intermediate results, new lists tend to be merged directly into a single, large, constantly expanding, combined list. This kind of process has already been identified as a source of inefficiency in discussing the behavior of the total elapsed time for a search as a function of memory size.

It may be possible to improve the efficiency of hardware utilization by waiting until several lists are available in core before doing any processing. However, the price of such an improvement may well be an increase in the total time required to complete the task.

5. CONCLUSION

A specialized processor for performing postings file access and coordination functions in inverted file retrieval systems has been presented. Design studies and simulation experiments indicate that the proposed system can be built using current technology and that it can process a complicated search in a large data base from 12 to 60 times faster than a large conventional computer. The speed-up is not as great for a short search involving only a few terms, but ten or more such searches can be processed concurrently with very little effect upon the system. In this way, the average elapsed time per search can be reduced drastically.

Application of the new system need not be restricted to information retrieval. It can be employed for any merging application, and in many cases it can be simplified considerably by the elimination of the coordination network.

While an exhaustive analysis of development costs is beyond the scope of this report, a fairly realistic estimate of component costs can be given since the subsystem designs presented in Chapter 2 are based on "off-the shelf" devices. Because semiconductor prices have been declining sharply in recent years, these figures should be regarded as conservative.

Table 5.1 lists several components and shows the number of units required and their approximate cost for the 16- and 256-path parallel systems. Hardware for a small system, excluding the control unit and the disk but including a 16K word × 32-bit 100ns memory, is estimated at about $50,000; a large unit would cost around $200,000. Addition of a control unit should not affect these numbers significantly.

Component	16-Paths		256-Paths	
	Number Required	Cost	Number Required	Cost
Gates	950	$ 360	28,000	$ 10,550
Latches	150	300	4,350	9,000
Flip-Flops	32	80	512	1,330
*Shift Registers	80	8,650	1,280	140,000
**Memory	1	39,600	1	39,600
Total		$48,990		$200,480

*32 bits

**16K words × 32 bits, 100ns cycle

Table 5.1. Component Cost Estimates

The Illiac IV head-per-track disk with a capacity of 10^9 bits and with the other capabilities assumed in this study cost approximately $500,000.

Mass storage alternatives under consideration include the various shift register technologies and the use of a moving head disk modified for parallel transmission from several tracks. If this last alternative proves attractive, a system containing a controller and eight drives comparable to the IBM 2314 could be obtained for about $70,000.

In summary, it appears that the hardware for a 16-path parallel retrieval system could be built for about $100K--$150K. A 256-path system would cost in the neighborhood of $1M.

Much remains to be done in the area of algorithm optimization and in the development of a theoretical basis for describing the performance of the system. Several unexpected phenomena have been observed in connection with the interaction between the disk and the hardware system, and a rigorous explanation for these is not yet available. Nevertheless, the system performs well and exhibits promise for extending the capabilities of inverted file retrieval systems.

LIST OF REFERENCES

[1] "Putting Law Libraries into Computers," Business Week, p. 36; January 26, 1974.

[2] Mead Data Central, Inc., Lexis, A Computer Based Legal Research Service (Undated Brochure).

[3] D. B. McCarn and J. Leiter, "On-Line Service in Medicine and Beyond," Science, pp. 318-324; July 27, 1973.

[4] W. R. Nugent, "The U. S. Patent Office Data Base: A Full Text Communications Format for Computer-Aided Classification, Retrieval and Examination of Patents," ASIS Proceedings, Vol. 8, pp. 179-184; 1971.

[5] K. E. Batcher, "Sorting Networks and Their Applications," Spring Joint Computer Conference, pp. 307-314; 1968.

[6] K. E. Batcher, "A New Internal Sorting Method," Goodyear Aerospace Corporation Report No. GER-11759; September 1964.

[7] K. E. Batcher, "Minimum-Time Merging Networks," Goodyear Aerospace Corporation Report No. GER-14122; December 1968.

[8] Motorola Semiconductor Products, Inc., MECL Integrated Circuits Data Book, Third Edition; 1973.

[9] Intel Data Catalog, Intel Corporation, Santa Clara, California; February 1973.

[10] Illiac IV Systems Characteristics and Programming Manual, IL4-PM1, Revision 4, Change 1; June 1970.

[11] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick and R. A. Stokes, "The Illiac IV Computer," IEEE Transactions on Computers, Vol. C-17, No. 8, pp. 746-757; August 1968.

[12] D. E. McIntyre, "An Introduction to the Illiac IV Computer," Datamation, Vol. 16, No. 4, pp. 60-67; April 1970.

[13] D. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," Proceedings of the IRE, pp. 1098-1101; September 1952.

[14] D. E. Knuth, The Art of Computer Programming, Vol. 3, Addison-Wesley, pp. 362-371; 1973.

[15] National Library of Medicine, Master MESH; November 1972.

VITA

William Howard Stellhorn was born in Fort Wayne, Indiana, in 1943. He received the B.S.E.E. and M.S.E.E. degrees from Purdue University in 1965 and 1966, respectively. His research activity at Purdue dealt with computer-aided analysis and synthesis of nonlinear networks. From 1967 to 1970 Mr. Stellhorn was employed as an Aerosystems Engineer with the General Dynamics Corporation in Fort Worth, Texas, where he developed and extended models for aircraft penetration studies. Since entering the University of Illinois in 1970, he has held a Research Assistantship in the Department of Computer Science. He is a member of the Association for Computing Machinery, the Institute of Electrical and Electronics Engineers, and Tau Beta Pi and an associate member of Sigma Xi.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-74-637	2.	3. Recipient's Accession No.
4. Title and Subtitle A SPECIALIZED COMPUTER FOR INFORMATION RETRIEVAL			5. Report Date October 1974 6.
7. Author(s) William Howard Stellhorn			8. Performing Organization Rept. No. UIUCDCS-R-74-637
9. Performing Organization Name and Address University of Illinois at Urbana-Champaign Department of Computer Science Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. US NSF GJ-36936
12. Sponsoring Organization Name and Address National Science Foundation Washington, D. C.			13. Type of Report & Period Covered Doctoral - 1974 14.

15. Supplementary Notes

16. Abstracts

Response time in large, inverted file document retrieval systems is determined primarily by the time required to access files of document identifiers on disk and perform the processing associated with a Boolean search request. This paper describes a specialized computer system capable of performing these functions in hardware. Using this equipment, a complicated sample search involving 70 terms and over 60,000 document references can be performed from 12 to 60 times faster than with a conventional machine, and many small searches can be processed concurrently with very little effect upon system performance.

A detailed description of the system, which can be realized with currently-available technology, is presented; and algorithms for controlling the progress of a search are discussed. Results from numerous simulations involving various system configurations and other factors are also reported.

17. Key Words and Document Analysis. 17a. Descriptors

Information Retrieval
Inverted Files
Parallel Processing
Merging
Computer Systems

17b. Identifiers. Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
RELEASE UNLIMITED	UNCLASSIFIED	115
	20. Security Class (This Page) UNCLASSIFIED	22. Price