RICE UNIVERSITY

DYNAMIC MEMORY INTERCONNECTIONS
FOR RAPID ACCESS

by

Balakrishna R. Iyer

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

APPROVED, THESIS COMMITTEE

James B. Sinclair

James B. Sinclair,
Assistant Professor of Computer Science
in Electrical Engineering, Chairman

Daniel Hirschberg

Daniel S. Hirschberg,
Assistant Professor of Computer Science
in Electrical Engineering

J. R. Jump

J. R. Jump,
Professor of Computer Science
in Electrical Engineering

HOUSTON, TEXAS
APRIL 1981

DYNAMIC MEMORY INTERCONNECTIONS FOR RAPID ACCESS

by

Balakrishna R. Iyer

ABSTRACT

A dynamic memory is a storage facility for fixed-size data items. The memory is comprised of cells, each cell capable of storing one datum. Data paths between cells are provided by a memory interconnection network. Each cell is directly connected to only a small number of cells. At every clock pulse, data items migrate from cell to cell via the data paths. The memory cells may be divided into several groups. A control mechanism provides each group of memory cells with a control signal. This control signal determines the data paths to be taken by data items contained in all cells within the group.

Many dynamic memory organizations have been proposed. These exhibit trade-offs between the time to access a datum randomly and the time to access serially a block of logically contiguous data. The access times for these organizations are derived where necessary and compared.

A new organization called the deck memory organization is proposed. Access times for the deck are determined and compared with access times derived for other organizations.

ACKNOWLEDGEMENTS

The entire work described in this thesis was carried out in the congenial environment of Rice University. I wish to express my gratitude to my thesis advisor Bart Sinclair for the uncountable hours he spent in shaping out this thesis and directing this research effort. My sincere thanks are also due to the two other members of my thesis committee Dan Hirschberg and Robert Jump who were lavish with their constructive criticism. I acknowledge the contributions of my colleagues and fellow graduate students Fary Attarzadeh, Danny Dias, John Dooley, M. Kannan, C. K. Krishnan, Manoj Kumar, Colin Pillay, Bapa Rao and Bhupendra Sheth through many interesting and thought provoking discussions.

And for their constant encouragement and unfailing support I find myself inadequate for words to express my feelings for my parents.

TABLE OF CONTENTS

Chapter 1	PRELUDE	1
1.1	A Model for a Dynamic Memory	1
1.2	The Dynamic Memory Access Problem	3
1.3	Simplistic Dynamic Memory Organizations	7
1.3.1	Linear Arrays	7
1.3.2	Two-dimensional Arrays	9
1.4	Thesis Outline	9
Chapter 2	THE STONE RANDOM ACCESS MEMORY	13
2.1	Introduction	13
2.2	Lower Bound for the Worst Case Access Time	13
2.3	Memory Interconnections	14
2.3.1	The Perfect Shuffle	14
2.3.2	The Shuffle Exchange	15
2.4	The Access Algorithm	17
Chapter 3	SURVEY OF OTHER DYNAMIC MEMORY ORGANIZATIONS	23
3.1	Introduction	23
3.2	Cyclically Interconnected Memories	24
3.2.1	The Aho-Ullman Memory	26
3.2.2	The Stone Cyclic Memory	28
3.2.3	The Two Permutation Wong-Tang Memory	31

3.2.4	The Three Permutation Wong-Tang Memory	33
3.2.5	The Lenfant Memory	36
3.3	Tour Memories	39
3.3.1	The Stone Tour Memory	39
3.3.2	The Fredrickson Extension	40
3.3.3	The Morris Generalization	40
3.4	Binary Tree Memories	41
3.5	Relative Performance	44
Chapter 4	THE DECK INTERCONNECTION NETWORK	46
4.1	Introduction	46
4.2	Structure of the Deck Memory	48
4.3	Access Times for the Deck Memory	50
4.4	A Design for Small Deck Memories	53
4.5	Advantages of Partitioning	56
Chapter 5	THE MULTI-LEVEL DECK MEMORY	57
5.1	Introduction	57
5.2	The Multi-level Deck Memory	57
5.3	Access Time Analysis	64
5.4	Heavily Loaded Multi-level Deck	66
5.4.1	Introduction	66
5.4.2	Read-Read Delay	68
5.4.3	Read-Write Delay	70
5.4.4	Write-Write Delay	72

5.4.5	Write-Read Delay	73
5.4.6	Expected Delay For The Deck Memory	75
Chapter 6	CONCLUSIONS	78
6.1	Performance Comparison	78
6.2	Extensions and Open Problems	81
6.3	Summary	83
Appendix A	The Aho-Ullman Memory	84
Appendix B	The Stone Cyclic Memory	92
Appendix C	The Two Permutation Wong-Tang Memory	96
Appendix D	The Three Permutation Wong-Tang Memory	103

LIST OF FIGURES

Figure 1.1	A Linear Array	8
Figure 1.2	A Two-dimensional Array	10
Figure 1.3	A Bubble Memory Organization	11
Figure 2.1	The Perfect Shuffle Permutation	16
Figure 2.2	The Shuffle Exchange Permutation	16
Figure 2.3	Access Algorithm for the Stone Random Access Memory	20
Figure 3.1	The Minus 1 Permutation	25
Figure 3.2	Permutations for the Aho-Ullman Memory	27
Figure 3.3	Permutations for the Stone Cyclic Memory	29
Figure 3.4	Permutations for the Two Permutation Wong- Tang Memory	32
Figure 3.5	The Plus 1 Permutation	35
Figure 3.6	Lenfant's Access Algorithm	38
Figure 3.7	Kluge's Memory Organization	43
Figure 4.1	Data Paths for a Deck Memory Organization	49
Figure 4.2	The Basic Structure of a Deck Memory	51
Figure 5.1	The Basic Structure of the Multi-level Deck Memory	58
Figure 5.2	The Control Propagation Network Organization	59

Figure 5.3	The Complete Multi-level Deck Organization	
.....		62
Figure 5.4	An Example of a Multi-level Deck Memory	65
Figure A.1	Access Algorithm for the Aho-Ullman Memory	
.....		86
Figure B.1	Access Algorithm for the Stone Cyclic Memory	
.....		94
Figure C.1	Access Algorithm for the Wong-Tang Two Per-	
mutation Memory		97

CHAPTER 1

PRELUDE

1.1.

A Model for a Dynamic Memory

With the evolution of computers, a variety of computer memories have flooded the computer market. A trade-off is observed between the speed and the size of memories. Small memories tend to be faster than large memories.

Fast, small memories in the form of registers and content-addressable memories are implemented in current semiconductor technology. Both semiconductor technology and magnetic core technology are used to implement primary memories. Much larger volumes of storage space are available on magnetic disks and drums, where data items are stored at fixed locations on a magnetic surface which rotates relative to a read/write mechanism. Data items stored at contiguous locations on the magnetic surface, are often grouped together into sets. Each such set of contiguous locations is known as a block. The average time to access a single datum on a rotating magnetic surface includes the time for one half rotation of the storage media, which is commonly several milliseconds [1]. A block of data can be accessed at rates comparable to primary memory access rates once the first datum of the block has been brought to the i/o port. Thus, disks and drums find their place naturally in the

storage hierarchy as relatively cheap mass memories which transfer blocks of data to and from primary memory.

The considerable gap of almost four orders of magnitude that exists between the access times for a datum in primary memories and in rotating mass memories may be bridged by emerging types of dynamic storage technologies such as magnetic domain and charge-transfer devices. In contrast to disks and drums, these devices require the continuous movement of data within the storage medium itself, and this movement need not be cyclic. Technological advances in recent years indicate that these new bulk storage devices may soon compete directly with conventional disks and drums. Integrated devices, consisting of magnetic bubbles [2], charge coupled devices or MOS shift registers, are already in limited use [3, 4].

The advent of new storage technologies requires the development of new memory models and access algorithms. One popular model of a dynamic memory due to Stone [3] forms the basis of the model used in this thesis. Current implementations of magnetic bubble memories and certain semiconductor memories show one or more characteristics of this model.

A dynamic memory is a storage facility for a set of fixed size data items. The memory consists of cells or locations, each storing one datum. Data paths are provided between these cells by a memory interconnection network. Each data path connects a source cell to a destination cell. Data items migrate constantly in the memory over these data paths. The number of paths leading out of a cell is

called the fanout of the cell. The fanout of a cell is always small, usually 2 or 3.

A control mechanism is provided for the memory. All access requests are made to this mechanism. The set of memory cells may be partitioned into several subsets. At every clock pulse the control mechanism provides each subset with a control signal. This signal determines the data path to be taken by the migrating datum currently residing at each memory cell in the subset. All data transfers start at the clock pulse. Data items move from their source cells to their destination cells in one clock period. A unit of time is defined as the interval between the initiation of two consecutive clock pulses. The data transfers within each subset, between clock pulses, constitute a permutation of the order of data items residing in that subset.

All data transfers between the memory and the outside world take place through a single memory cell called the i/o port or access window. The contents of the access window can be read and written from outside the memory.

1.2. The Dynamic Memory Access Problem

Suppose t_1 is the time at which the control mechanism starts to service a request to read a datum from the memory. Let t_2 be the time ($t_2 \geq t_1$) at which the requested datum is available at the i/o port. The time difference $t_2 - t_1$ is defined as the random read

access time for the datum. The random write access time is defined similarly, with t_1 being the time when the control starts servicing a write request and t_2 the time when the datum is actually written into the i/o port of the memory. When the random read and write access times are the same, they are referred to simply as the random access time.

A distinction needs to be made between the logical addresses of data and the physical addresses of locations, since data migrate in the memory. A block of data is defined as a set of b logically contiguous data items. A block access consists of the access to the datum at the head of the block followed by the access of each of the remaining $b-1$ logically contiguous data items of the block, in the order of their logical addresses.

Let t_1 be the time at which the control starts servicing a block access request. For a read request, let t_2 be the time at which the last datum of the block is read from the i/o port. For a write request, let t_2 be the time at which the last datum of the block is written into the i/o port. The time difference $t_2 - t_1$ is called the read or write block access time.

The random and block access times defined above are critical parameters for situations in which the memory traffic is light. Each arriving request is likely to find the memory ready to service the request. In situations in which there is heavy memory traffic,

arriving access requests are likely to find the memory servicing a prior access request, with some activity (a read or a write) scheduled at the i/o port. For such situations it becomes meaningful to define the term delay.

Consider the presence of two consecutive access requests, which have yet to receive any service from the memory. The access requests are satisfied on a first-come first-served basis. Let t_1 be the time when data transfers through the i/o port for the first memory request terminate and let t_2 be the time at which data transfers through the i/o port for the second memory request begin. The time interval $t_2 - t_1$ is defined to be the delay between the two requests. Several factors contribute to the delay. In some memory organizations, termination of activity at the i/o port must be followed by a time period during which the memory re-orders the data internally, before the next request can be serviced. This time and the time to access the first datum of the second block contribute towards the delay. In other memory organizations consecutive access requests can be partially overlapped, thus reducing the delay. The throughput of the memory is the data transfer rate between the memory and the outside world. The average throughput of the memory is related inversely to the average delay.

The dynamic memory access problem is to design a dynamic memory that functions efficiently when subjected to both heavy and light memory traffic. One part of the problem is to make random access

times small. However, computer memories are often subject to block accesses. Thus, there is also a need to minimize block access times. Trade-offs must be made in satisfying these two goals. In addition, the memory should exhibit small delays.

Two major approaches may be taken in solving the dynamic memory problem. In one approach one considers permissible memory interconnections based on the capabilities of current technology, and searches for good access mechanisms for the memory. This approach is appropriate for rigidly structured memories such as magnetic bubble memories. An example is the work of Wong and Coppersmith [5], who obtained access times linear in the number of data items in the memory.

The other major approach is to disregard the limitations of current technology and investigate memory interconnection structures which permit the implementation of fast access techniques. The feasibility of implementing the required interconnection and control structure, being basically a technological question, is left as an open question. This thesis and the work surveyed herein take this latter approach.

The distinction made between the logical address of a datum and the physical address of the location where it resides gives rise to the storage mapping function $s()$, also referred to as the memory map. $s(i)$ is defined to be the physical location currently containing datum i . The initial memory map is defined as the memory map at $t = 0$, i.e., when the memory first becomes operative. Throughout this

thesis it is assumed that the initial memory map is $s(i) = i$, for all logical addresses i .

An access algorithm is a procedure that determines the sequence of control signals to be applied to the memory in order to access a particular datum. The control signals determine which permutations are to be applied to subsets of memory cells. Access algorithms in effect manipulate the memory map.

1.3. Simplistic Dynamic Memory Organizations

In this section two simplistic dynamic memory organizations are examined. Throughout this thesis N denotes the number of distinct data items that can be stored in the memory.

1.3.1. Linear Arrays

If the fanout of each cell is restricted to 1 then the only interconnection of N cells that permits access to each cell is the cyclic interconnection shown in Figure 1.1. The memory cells are labeled from 0 through $N-1$. There is a data path from cell i to cell $i \ominus 1$, where $\ominus$ represents subtraction modulo N . Cell 0 is designated as the i/o port. The random access time is the number of links to be traversed by a datum to reach cell 0. The random access time varies between 0 and $N-1$. The average random access time is $\frac{N-1}{2}$. The $O(N)$ access time becomes unacceptable as the size of the memory increases.

Linear arrays are of use in small hardware-assisted data bases. They are also attractive from layout considerations. It is easy to

A Linear Array

Figure 1.1

construct such an interconnection on an integrated chip using repetitive logic.

1.3.2. Two-dimensional Arrays

The linear structure of the previous section can be extended to two dimensions as shown in Figure 1.2. N is assumed to be a perfect square. The N memory locations are labeled from $(0,0)$ through $(\sqrt{N}-1, \sqrt{N}-1)$. Location $(0,0)$ is the i/o port. The fanout of each cell is 2. Data paths from cell (i,j) lead to cells $(i+1,j)$ and $(i,j+1)$. The random access time can be shown to be $O(\sqrt{N})$.

A bubble memory organization described by Bhandarkar and Juliussen [2], shown in Figure 1.3, can be considered as an approximation to the two-dimensional array.

1.4. Thesis Outline

Both of the above organizations have poor random access times as compared to the organizations proposed in this thesis, particularly as N becomes large. The rest of this thesis examines various designs for dynamic memories that are intended to reduce access times. Chapter 2 contains a characterization of a memory organization proposed by Stone [3], that minimizes random access times. Chapter 3 is a survey of other dynamic memory organizations proposed in the literature. The basic structure of the memory organization proposed in this thesis is described in Chapter 4. A design for large memories based on this structure is considered in Chapter 5. Comparisons of the performance of this memory organization with other

A Two-dimensional Array

Figure 1.2

A Bubble Memory Organization

Figure 1.3

organizations surveyed are made in Chapter 6.

CHAPTER 2

THE STONE RANDOM ACCESS MEMORY

2.1. Introduction

Stone [3] shows that the lower bound for random access time in a dynamic memory is logarithmic in the number of locations in the memory. He proposes an organization for a memory of N cells storing N data items. Memory locations are numbered from 0 through $N-1$. Location 0 is the i/o port for the entire memory. The cells are considered to constitute a single set and a single binary-valued control signal triggers a permutation of the contents of the memory once every time unit. Two permutations are defined for the memory. Cell fanout is 2.

In this chapter a bound on access time for this memory organization is examined. A notation is developed to define the proposed permutations and provide a basis for understanding the access algorithm.

2.2. Lower Bound for the Worst Case Access Time

A lower bound for the access time can be determined by an information theoretic argument. The following observations are made about the Stone organization. The datum in the i/o port is available immediately, on request. After one time unit, during which one of two permutations has been applied, one of at most two different data

items will have been made available at the i/o port. One of at most four new data items will have been brought to the i/o port after one additional time unit. Extending this argument, and summing over time units 0 through k demonstrates that for a worst case access time of k time units, at most $2^{k+1} - 1$ data items can be contained in the memory. Therefore, for a memory of N locations, the worst case access time is bounded from below by $\lfloor \log(N+1) \rfloor^1 - 1$.

2.3. Memory Interconnections

Stone [3] considers memories of $N = 2^m$ locations, where m is a positive integer. The access algorithm for the memory is considerably simplified by this restriction. At the application of a permutation on the memory, the datum in location i is routed to a destination $d(i)$. The two permutations defined are called the Perfect Shuffle and the Shuffle Exchange permutations.

2.3.1. The Perfect Shuffle

The perfect shuffle (or simply the shuffle) is analogous to a shuffle of a pack of cards [6]. The shuffle permutation d_s is defined for even values of N by the following mapping:

$$\begin{aligned} d_s(i) &= 2i && \text{for } 0 \leq i \leq N/2 - 1 \\ &= 2(i-N/2) + 1 && \text{for } N/2 \leq i \leq N - 1 \end{aligned}$$

¹Unless otherwise specified all logarithms in this thesis are to the base 2.

The perfect shuffle permutation is illustrated in Figure 2.1 for $N = 8$. Each memory element has been drawn twice to illustrate the source and destination of an interconnection path.

Stone [6] shows that if $N = 2^m$ and $i = \sum_{k=0}^{m-1} b_k * 2^k$, where b_k is either 0 or 1 for $0 \leq k < m$, then $d_s(i) = b_{m-1} * 2^0 + \sum_{k=0}^{m-2} b_k * 2^{k+1}$. The binary representation of $d_s(i)$ is the left cyclic shift of the binary representation of i . $d_s(d_s(i))$ can be written as $d_s^2(i)$. Similarly $d_s(d_s(\dots(d_s(i))\dots))$, the result of p such applications of the shuffle permutation, is written as $d_s^p(i)$. Note that $d_s^m(i) = i$.

2.3.2. The Shuffle Exchange

The shuffle exchange permutation d_{sx} can be considered as a composition of two permutations, a shuffle permutation followed by an exchange permutation. An exchange consists of an interchange of data items between locations whose physical addresses differ only in the least significant bit. The datum in location j is routed to location j^* , where j^* is obtained by complementing the least significant bit of j . The shuffle exchange is illustrated for $N = 8$ in Figure 2.2.

Let j be an integer such that $0 \leq j < 2^m$. $[j_{m-1} \dots j_1 j_0]$ is the binary representation of j . Then $j \oplus [b_{m-1} \dots b_1 b_0]$ is defined to be the m -bit binary integer $[j_{m-1} \oplus b_{m-1} \dots j_1 \oplus b_1 \ j_0 \oplus b_0]$, where $\oplus$ represents the exclusive or operation. A data item in a location

The Perfect Shuffle Permutation

Figure 2.1

The Shuffle Exchange Permutation

Figure 2.2

with physical address $j \oplus [b_{m-1}b_{m-2} \dots b_1b_0]$ is routed by the exchange permutation to a location with physical address $j \oplus [b_{m-1}b_{m-2} \dots b_1\hat{b}_0]$, where $\hat{b}_0$ is the complement of b_0 .

Using this notation, the shuffle permutation can be represented as

$$d_s(j \oplus [b_{m-1}b_{m-2} \dots b_1b_0]) = d_s(j) \oplus [b_{m-2}b_{m-3} \dots b_0b_{m-1}].$$

A shuffle exchange permutation, being a shuffle followed by an exchange, is described by

$$d_{sx}(j \oplus [b_{m-1}b_{m-2} \dots b_1b_0]) = d_s(j) \oplus [b_{m-2}b_{m-1} \dots b_0\hat{b}_{m-1}].$$

2.4. The Access Algorithm

Using the notation of the previous section, the initial memory map is $s(i) = d_s^0(i) \oplus [00 \dots 00]$. It will be shown that, at any time, the memory map can be written in the form $s(i) = d_s^p(i) \oplus [b_{m-1}b_{m-2} \dots b_1b_0]$, such that $0 \leq p < m$. Hence the number p and the bit vector $[b_{m-1}b_{m-2} \dots b_1b_0]$ completely characterize the memory map.

Assume that $s(i)$ can be represented as $d_s^p(i) \oplus [b_{m-1}b_{m-2} \dots b_1b_0]$.

Application of the shuffle permutation routes the contents of location $s(i)$ to location $d_s(s(i))$. The new memory map $s(i) =$

$$d_s(s(i)) = d_s(d_s^p(i) \oplus [b_{m-1}b_{m-2} \dots b_1b_0]) = d_s^{p+1}(i) \oplus [b_{m-2}b_{m-3} \dots b_0b_{m-1}].$$

Since $d_s^m(i) = i$, the addition $p+1$ can be considered as

modulo m addition. The shuffle exchange permutation is a composition of the shuffle and the exchange permutation. Hence the effect of the shuffle exchange permutation on the memory map is to transform it into $s(i) = d_s^{p+1}(i) \oplus [b_{m-2}b_{m-3} \dots b_0\hat{b}_{m-1}]$, where the addition is modulo m . In terms of the memory map $s(i) = d_s^p(i) \oplus [b_{m-1}b_{m-2} \dots b_1b_0]$

$$d_s \text{ maps } \begin{cases} p \text{ into } p+1 \text{ mod } m \\ [b_{m-1}b_{m-2} \dots b_1b_0] \text{ into } [b_{m-2}b_{m-3} \dots b_0b_{m-1}] \end{cases}$$

$$d_{sx} \text{ maps } \begin{cases} p \text{ into } p+1 \text{ mod } m \\ [b_{m-1}b_{m-2} \dots b_1b_0] \text{ into } [b_{m-2}b_{m-3} \dots b_0\hat{b}_{m-1}] \end{cases}.$$

Assume that the memory map is given by

$$s(i) = d_s^p(i) \oplus [b_{m-1}b_{m-2} \dots b_1b_0], \text{ when an access request for datum } j$$

is encountered by the memory. The current location of datum j is

$$s(j) =$$

$$[j_{m-p-1}j_{m-p-2} \dots j_{m-p+1}j_{m-p}] \oplus [b_{m-1}b_{m-2} \dots b_1b_0]$$

$$= [j_{m-p-1} \oplus b_{m-1} \ j_{m-p-2} \oplus b_{m-2} \dots j_{m-p+1} \oplus b_1 \ j_{m-p} \oplus b_0]. \text{ The access}$$

algorithm generates a sequence of shuffle and shuffle exchange

permutations which, when applied to the memory, route datum j from

location $s(j)$ to the i/o port (at location 0). The shuffle routes

datum j into the location whose address, in binary representation, is

$$[j_{m-p-2} \oplus b_{m-2} \ j_{m-p-3} \oplus b_{m-3} \dots j_{m-p} \oplus b_0 \ j_{m-p-1} \oplus b_{m-1}] =$$

$$[j_{m-p-2}j_{m-p-3} \dots j_{m-p}j_{m-p-1}] \oplus [b_{m-2}b_{m-3} \dots b_0b_{m-1}] =$$

modulo m addition. The shuffle exchange permutation is a composition of the shuffle and the exchange permutation. Hence the effect of the shuffle exchange permutation on the memory map is to transform it into $s(i) = d_s^{p+1}(i) \oplus [b_{m-2}b_{m-3} \dots b_0\hat{b}_{m-1}]$, where the addition is modulo m . In terms of the memory map $s(i) = d_s^p(i) \oplus [b_{m-1}b_{m-2} \dots b_1b_0]$

$$d_s \text{ maps } \begin{cases} p \text{ into } p+1 \text{ mod } m \\ [b_{m-1}b_{m-2} \dots b_1b_0] \text{ into } [b_{m-2}b_{m-3} \dots b_0b_{m-1}] \end{cases}$$

$$d_{sx} \text{ maps } \begin{cases} p \text{ into } p+1 \text{ mod } m \\ [b_{m-1}b_{m-2} \dots b_1b_0] \text{ into } [b_{m-2}b_{m-3} \dots b_0\hat{b}_{m-1}] \end{cases}.$$

Assume that the memory map is given by

$s(i) = d_s^p(i) \oplus [b_{m-1}b_{m-2} \dots b_1b_0]$, when an access request for datum j is encountered by the memory. The current location of datum j is

$$\begin{aligned} s(j) &= [j_{m-p-1}j_{m-p-2} \dots j_{m-p+1}j_{m-p}] \oplus [b_{m-1}b_{m-2} \dots b_1b_0] \\ &= [j_{m-p-1} \oplus b_{m-1} \ j_{m-p-2} \oplus b_{m-2} \dots j_{m-p+1} \oplus b_1 \ j_{m-p} \oplus b_0]. \end{aligned}$$

The access algorithm generates a sequence of shuffle and shuffle exchange permutations which, when applied to the memory, route datum j from location $s(j)$ to the i/o port (at location 0). The shuffle routes datum j into the location whose address, in binary representation, is $[j_{m-p-2} \oplus b_{m-2} \ j_{m-p-3} \oplus b_{m-3} \dots j_{m-p} \oplus b_0 \ j_{m-p-1} \oplus b_{m-1}] =$
 $[j_{m-p-2}j_{m-p-3} \dots j_{m-p}j_{m-p-1}] \oplus [b_{m-2}b_{m-3} \dots b_0b_{m-1}] =$

Datum to be accessed has logical address j .

Given :

$$s(i) = d_s^p(i) \oplus [b_{m-1} \dots b_1 b_0].$$

$$B = [b_{p-1} \dots b_0 b_{m-1} \dots b_p].$$

$2*C$ represents a logical left shift operation.

$\oplus$ denotes the exclusive or operation.

Access Algorithm for the Stone Random Access Memory

Figure 2.3

comparison of bits j_{m-p-1} and b_{m-1} is done by ANDing the contents of the A-register with the C-register and comparing it with the result of ANDing the B-register with the C-register.

There may be periods during which no access request need be serviced by memory. During these periods either the shuffle or the shuffle exchange permutation may be applied to the memory. Register B needs to be updated with application of each memory permutation to contain the logical address of the datum currently in the i/o port. When a shuffle permutation is applied to the memory, the contents of the B-register are shifted left logically one position; for a shuffle exchange permutation, the B-register is subject to a logical left shift followed by the complementation of its least significant bit.

Each access requires $\log N$ time units, since the algorithm looks at each bit in the address. The average and worst case random access times are $\log N$. A block access for a block of b data items takes $b \log N$ time. The memory organization described in this chapter will be referred to as the Stone random access memory organization.

Stone [3] has indicated that certain savings in the access time can be obtained if, after application of every permutation, the algorithm checks to see if datum j has reached the i/o port and stops generating permutations when it has. This would leave the memory map $s(i)$ as $d_s^{p'}(i) \oplus [j_{m-p'-1} \dots j_0 j_{m-1} \dots j_{m-p}]$, for some p' . For such a modified organization the worst case access time is still $\log N$, but the average access time is reduced to $\log N - 1$. The

organization proposed in this thesis will not make use of the savings due to such a modification.

CHAPTER 3

SURVEY OF OTHER DYNAMIC MEMORY ORGANIZATIONS

3.1. Introduction

This chapter is a brief summary of various attempts to solve the dynamic memory problem posed in Chapter 1. Three different approaches have been used. The memory organizations proposed in the literature will be classified according to the underlying approach. Stone's organization [3] of the dynamic memory, though exhibiting good random access behavior, does not give good block access times. Aho and Ullman indicate [7] that implementing a cyclic interconnection within the memory gives good block access properties. After initial accesses for a block of sequential data, this approach permits unit access time per data item for the rest of the block. Four organizations, due to Aho and Ullman [7], Stone [8], Wong and Tang [9, 10] and Lenfant [11], that follow this approach are summarized in this chapter.

In another approach to providing good block access characteristics, Stone [8] makes the observation that, given only the two permutations shuffle and shuffle exchange, it is possible to construct a sequence of permutations that brings each datum into the i/o port just once. The length of this sequence is the same as the number of locations in the memory. If the logical addresses of data are so recoded that the sequence in which data visit the i/o port is

in the order of their coded addresses, block access takes unit time per access after the first access of the block. Such sequences are called tours. Memories that rely on such tour structures are defined to be tour memories. Contributions towards the understanding of tour memories made by Stone [8], Fredrickson [12] and Morris, Valiere and Wisniewski [13] are briefly discussed in this chapter.

The third approach has been prompted by practical considerations of layout of the memory cells and the interconnection structure on a planar surface. Kluge's work [1], reflecting this approach, describes a planar design of a memory that exhibits logarithmic random access time. The average rate at which data items are accessed during a block access is close to one datum per time unit. This organization is also described in this chapter.

3.2. Cyclically Interconnected Memories

Cyclically interconnected memories are characterized by the presence of the minus 1 permutation d_c , illustrated in Figure 3.1. If the data items are arranged in the memory locations sequentially by their logical addresses, then use of the cyclic interconnection enables data items to be accessed once every time unit starting with the access to the datum in the i/o port. The minus 1 permutation also plays an active role in the random access of data. Different implementations of cyclically interconnected memories are examined in the following sub-sections.

Number of cells = N

The Minus 1 Permutation

Figure 3.1

3.2.1. The Aho-Ullman Memory

In the Aho-Ullman memory organization [7], the total number of locations in the memory is $N = 2^m - 1$, with locations numbered 0 through $N-1$. Location 0 is the i/o port. Besides the minus 1 permutation (d_c), the memory implements the shuffle permutation defined for the $2^m - 1$ locations by d_s ,

$$d_s(j) = 2j \bmod N.$$

For $N = 7$, the two permutations d_s and d_c are illustrated in Figure 3.2.

Aho and Ullman [7] propose an access algorithm which permits random access from the memory in worst case time of $2 \log(N+1) - 2$ and for large-sized memories, an average access time of $1.5 \log(N+1) - 2$. The access algorithm is described in Appendix A. A block access starts with a random access to the first datum of the block. Access of the next datum requires between 1 and $\log_2(N+1)$ time. After the access of the first two data items of the block, access proceeds at the rate of one datum per time unit.

Aho and Ullman generalize this organization to memories of size $N = r^m - 1$. The permutations implemented are d_c and d_s^r , where

$$d_s^r = ri \bmod N.$$

The organization of this memory is described in Appendix A. Access times are also evaluated. The value of r can be chosen to minimize access times, with $r = 3$ giving the smallest worst case random access

Permutations for the Aho-Ullman Memory

Figure 3.2

time of $1.893 \log(N+1) - 2$. The minimum worst case block access time, for a block of b items ($b \geq 2$), is $2.5 \log(N+1) + b - 4$ for $r=4$.

The choice of $r = 4$ minimizes the average random access time to $1.25 \log(N+1) - 1.333$ and also minimizes the average block access time to $1.25 \log(N+1) + b - 1$.

3.2.2. The Stone Cyclic Memory

The Stone cyclic memory [8] is a minor modification of the Aho-Ullman memory. The memory consists of $N = 2^m - 1$ cells, numbered from 1 to N (note the difference in numbering). Location 1 is the i/o port. Besides the minus 1 permutation (d_c) the memory has the shuffle permutation defined for the $2^m - 1$ locations by

$$d_s(i) = 2i \bmod N \text{ for } 1 \leq i < N$$

$$d_s(N) = N.$$

The two permutations d_s and d_c are illustrated in Figure 3.3, for $N = 7$.

The access algorithm proposed by Stone is given in Appendix B. The worst case random access time for the memory is $2 \log(N+1)$ and the average case is $1.5 \log(N+1)$. Subsequent access to logically consecutive data items requires unit time per access. The advantage of this memory organization lies in the simplicity of its access algorithm. While the Aho and Ullman access algorithm involves the

Permutations for the Stone Cyclic Memory

Figure 3.3

addition of two m -bit integers in each iteration with possible carry propagation delay, the Stone access algorithm replaces this addition by bit comparison and cyclic shifts. The worst and average case block access times for a block of b ($b \geq 2$) data items for the Stone cyclic memory are $2 \log(N+1) + b - 1$ and $1.5 \log(N+1) + b - 1$ respectively.

Stone suggests [8] that this memory organization can be generalized to memories of size $N = r^m - 1$, with the implementation of the permutations d_c and d_s^r , where

$$d_s^r(i) = ri \bmod N \text{ for } 1 \leq i < N$$

$$d_s^r(N) = N$$

This generalized organization is described in Appendix B. The access times are also evaluated and optimized with respect to the choice of r . The worst case random and block access times are minimized to $1.893 \log(N+1)$ and $1.893 \log(N+1) + b - 1$ respectively, for the choice of $r = 3$. Thus the Stone cyclic memory has better worst case block access performance when compared to the Aho-Ullman memory. The minimum average case random access time and block access times are minimized to $1.25 \log(N+1)$ and $1.25 \log(N+1) + b - 1$ respectively, for $r = 4$. There is little difference between average access times for the Stone cyclic memory and the Aho-Ullman memory.

3.2.3. The Two Permutation Wong-Tang Memory

Wong and Tang propose [9, 10] another class of memory organizations which can be viewed as a generalization of the Aho-Ullman memory organization. The number of locations in this organization is $N = 2^m - 1$. Wong and Tang note that m can be written as a product of two integers c and w , $m = cw$. Members of a class are distinguished by the value of w . Memory locations are labeled from 0 through $N-1$. Location 0 is the i/o port. Besides the minus 1 permutation (d_c), the multiple shuffle permutation $d_{s,w}$ is defined for the $2^m - 1$ memory locations by

$$d_{s,w}(i) = [2^{(c-1)w} i] \bmod N \text{ for } 0 \leq i < N.$$

The two permutations are illustrated in Figure 3.4, for $N=15$, $w=2$, $c=2$.

Appendix C contains a description of the access algorithm proposed by Wong and Tang. Random access requires time $\frac{2^w}{w} \log(N+1) - 2$ in the worst case and $\frac{2^{w+1}}{2w} \log(N+1) - \frac{2^w}{2^w-1}$ in the average case. A block access starts with a random access to the first datum of the block. It takes between 1 and $\frac{\log_2(N+1)}{w}$ time units to access the next datum. After the access of the first two data items of the block every successive datum is accessed in unit time. The worst case block access time is $\frac{2^{w+1}}{w} \log(N+1) + b - 2$. The average block access time is $\frac{2^{w+1}}{2w} \log(N+1) + b - 1$. The average block access time

Permutations for
the Wong-Tang Two Permutation Memory

Figure 3.4

is derived in Appendix C. These times are optimized over different values of w .

For a dynamic memory of size $N = r^m - 1$, Wong and Tang generalize the multiple shuffle permutation (see Appendix C) to $d_{s,w}^r$ given by

$$d_{s,w}^r(i) = [r^{(c-1)w}i] \bmod N \text{ for } 0 \leq i < N.$$

The optimization of access times can be carried out over different values of r . Table 3.1 summarizes the results of this optimization.

3.2.4. The Three Permutation Wong-Tang Memory

Wong and Tang [9, 10] add another cyclic permutation d_a to the two permutation Wong-Tang memory. This modified organization is referred to as the three permutation Wong-Tang memory.

Wong and Tang propose a class of memory organizations with $N = 2^m - 1$ cells, where $m = cw$, as described in the previous section. Memory locations are labeled from 0 through $N-1$. Location 0 is the i/o port. The additional permutation d_a is defined by

<u>Access Time</u>	<u>Minimum Access Time</u>	<u>Memory Organization</u>
Worst Case Random	$1.893 \log(N+1) - 2$	$r=3, w=1$
Avg. Case Random	$1.25 \log(N+1) - 1.333$	$r=2, w=2$ or $r=4, w=1$
Worst Case Block	$2.5 \log(N+1) + b - 2$	$r=2, w=2$ or $r=4, w=1$
Avg. Case Block	$1.25 \log(N+1) + b - 1$	$r=2, w=2$ or $r=4, w=1$

Table 3.1
Access Time Minimization for the Two Permutation Wong-Tang Memory
(Block accesses are for b data items)

$$d_a(i) = (i+1) \bmod N \text{ for } 0 \leq i < N$$

The permutation d_a , which will also be referred to as the plus 1 permutation, is illustrated in Figure 3.5, for $N=7$. Wong and Tang construct an access algorithm that uses the three permutations d_c , d_a and $d_{s,w}$. They evaluate and minimize the random access time for different organizations in the class. The block access time is analyzed in Appendix D and similarly minimized. Wong and Tang generalize this memory organization to radix r in a manner similar to the generalization of the two permutation Wong-Tang memory. This generalization uses permutations d_c , d_a and $d_{s,w}^r$, and is described in Appendix D. Analysis of worst and average case access times is shown. Average block access times are also evaluated in Appendix D. The access times are minimized for different members of this class of organization. Table 3.2 shows the results for optimum values of r and w .

<u>Access Time</u>	<u>Minimum Access Time</u>	<u>Memory Organization</u>
Worst Case Random Access	$1.2621 \log(N+1) - 1$	$r=3, w=1$
Avg. Case Random Access	$0.967 \log(N+1) - 1.2$	$r=6, w=1$
Worst Case Block Access	$1.723 \log(N+1) + b - 3$	$r=5, w=1$
Avg. Case Block Access	$0.967 \log(N+1) + b - 2.333$	$r=6, w=1$ or $r=4, w=1$

Table 3.2
Access Time Minimization for the Three Permutation Wong-Tang Memory
(Block Accesses are for b data items)

Number of cells = $N = 7$

The Plus 1 Permutation

Figure 3.5

3.2.5. The Lenfant Memory

Lenfant [11] considers a memory of N cells storing N data items. N can be any positive integer value. Logical and physical addresses lie in the range $[0, N-1]$; location 0 is the i/o port. As in the Stone cyclic memory, data items are sequentially arranged in memory before and after every access. Thus each random access can be thought of as a logical cyclic shift of contents of the memory. Block access is efficient due to the unit cyclic shift (d_c) permutation.

The memory is not limited to two permutations. Addresses are represented in mixed radix base. If $B = (R_0, R_1, \dots, R_{m-1})$ is a sequence of positive integers each greater than 1, then the sequence of numbers $b_{m-1}, b_{m-2}, \dots, b_0$, where $0 \leq b_i < R_i$ for $0 \leq i < m$, is a number in mixed base B with value $\sum_{i=0}^m b_i G_i$, where $G_0 = 1$ and $G_i = R_{i-1} G_{i-1}$ for $1 \leq i \leq m$. If $\{Z_1, Z_2, \dots, Z_s\}$ is the set of distinct integers in B , then there are s additional permutations d_{Z_k} defined for the memory, where

$$d_{Z_k}(j) = Z_k j \bmod N$$

$$\begin{aligned} &\text{for } 0 \leq j < N \\ &\text{and } 1 \leq k \leq s. \end{aligned}$$

The permutation d_{Z_k} is called a Z_k -shuffle. Lenfant ensures that these are indeed permutations by restricting B to sequences that

satisfy $R_0 R_1 \dots R_{m-1} = 1 \pmod{N}$.

The access algorithm proposed by Lenfant is given in Figure 3.6. For a memory of size N the average random access time can be optimized over the mixed radix, B . Since there are infinitely many sequences that are candidates for the mixed radix B , Lenfant confines his optimization to those sequences that are prime factors of $qN + 1$, where $1 \leq q \leq 20,000$. Furthermore, B 's for which the number of distinct R_i 's is greater than 5 are excluded from consideration. This limits the number of permutations defined for a memory to 6. Lenfant carries out this optimization for several sizes of memory between 32 and 8192.

From the optimization data for memory of sizes 2^k , Lenfant makes the empirical observation that the minimum average random access time lies between $1.24 \log N$ and $1.26 \log N$. There is one exception, the case where $N = 2^{12} - 1$, for which the average random access time is approximately $1.46 \log_2(4096)$.

A slightly different model of the dynamic memory is examined by Lenfant, where the shuffle takes 5 time units while the d_c permutation takes one. For memories of size 2^k , the minimum average random access time is between $2.84 \log N$ and $2.93 \log N$. For memory sizes of $2^k - 1$, the minimum average access time is between $2.86 \log N$ and $2.99 \log N$.

Given :

Logical address of datum to be accessed : j

Logical address of contents of the i/o port : i

$d = j - i = \{d_1\}$ in the mixed base $B = \{R_1\}$.

Lenfant's Access Algorithm

Figure 3.6

3.3. Tour Memories

Another approach to the problem has as its basis the existence of special access sequences called tours [8]. In this approach one seeks permutations specifically suited for random access of data. Sequences of these permutations are then sought that bring each datum into the i/o port only once. Addresses are recoded so that the sequence in which data passes through the i/o port is in increasing order of the addresses. Stone [8], Fredrickson [12] and Morris et. al. [13] use this approach. Stone proposes the idea of the tour memory for memories that consist of N cells where N is restricted to specific powers of 2. Locations are numbered from 0 to $N-1$. Location 1 is the i/o port. Fredrickson extends the idea of tour memories by considering various other locations in the memory to be the i/o port. Morris, et. al., generalize the size of the tour memory to N , not restricted to be a power of 2. They show the existence of tour memory organizations for various sizes of memories.

3.3.1. The Stone Tour Memory

The Stone tour memory [8] consists of 2^m locations addressed in the range $[0, 2^m-1]$. The permutations implemented in the memory are the shuffle and the exchange shuffle permutations. Location 1 is the i/o port. Stone suggests the construction of a sequence of allowable permutations that brings the contents of each location in the memory to the i/o port exactly once. Such a sequence is called a tour. Since location 1 is the i/o port these tours are called tours at location 1. Tours at location 1 are shown to exist for $N = 4, 8$, and

16. The average random access time for large memories is $1.5 \log N$ and the worst case time is $2 \log N - 1$. An address recoding scheme permits access to logically consecutive data items at the rate of one access per time unit.

3.3.2. The Fredrickson Extension

In an unpublished note [12], Fredrickson makes the observation that it is not necessary to stipulate location 1 as being the i/o port. Thus, with the flexibility of choice of the i/o port over the address range, tours are shown to exist for other sizes of memory. He also suggests a two-dimensional organization of memory to take advantage of tour structures in two dimensions.

3.3.3. The Morris Generalization

Morris et. al. [13] generalize the definition of the shuffle and the shuffle exchange permutations to arbitrary size memories. The Stone random access memory is generalized to arbitrary N . The worst case random access time for such memories is shown to be $\lceil \log N \rceil$ time.

In investigating the theoretical properties of tours, they derive necessary conditions for the existence of tours for any designation of the i/o port. Certain ad hoc procedures are demonstrated that construct tours for larger memories given tours for smaller sized memories. Existence of tours for memories of size 2^r for $2 \leq r \leq 6$ is demonstrated.

No general theory exists to assure the existence of tours for arbitrary memory sizes. Furthermore, tour memories require address recoding which is likely to affect the scheme adversely in terms of both access time and layout space.

3.4. Binary Tree Memories

Kluge [1] proposes a dynamic memory organization with two non-cyclic permutations. He considers a memory of size $N = 2^m - 1$, numbering the locations from 1 through N . Location 1 is the i/o port. Two permutations are defined on the memory as follows

$$\text{For } 2^{2h} \leq i < 2^{2h+1}$$

$$\text{and } 0 \leq h < \frac{m}{2}$$

$$d_A(i) = 2i + 1$$

$$d_A(2i+1) = 2i$$

$$d_A(2i) = i$$

$$d_A(i) = i \text{ for all other } i \text{ such that } 1 \leq i \leq N.$$

$$\text{For } 2^{2h-1} \leq i < 2^{2h}$$

$$\text{and } 1 \leq h < \frac{m}{2}$$

$$d_B(i) = 2i + 1$$

$$d_B(2i+1) = 2i$$

$$d_B(2i) = i$$

$$d_B(i) = i \text{ for all other } i \text{ such that } 1 \leq i \leq N.$$

The memory and the permutations are shown for $m=5$ are shown in Figure 3.7. The interconnection structure is a tree with some additional interconnection. Kluge gives an algorithm that generates an access sequence for random access within the memory. The worst case random access time is shown to be $2 \log(N+1)$. In Figure 3.7 this corresponds to the time taken to bring the item in location 31 to the i/o port.

Access to the cell at depth 0 takes 0 time. As the depth of a cell increases by 1 the average access time increases by 1.5 units.

$$\text{Thus the average access time is } \frac{1.5 \sum_{i=0}^{m-1} i 2^i}{2^m - 1} = \frac{3((m-2)2^{m-1} - (m-1)2^{m-2} + 1)}{2^m - 1}.$$

For large values of m this is approximately $1.5 \log(N+1) - 4.5$.

Kluge gives an algorithm for accessing a block of b items in memory. This access sequence delivers data in spurts of 2 data items, unit time apart. Between these spurts the memory executes at least two permutations. The time to access all 2^g items at the same depth is $3(2^g - 1)$. Thus, an average of 3 time units is spent to access every data item. When the memory is partitioned into two modules and addresses interleaved, the access time for these 2^g data items is reduced to $3(2^{g-1} - 1)$, for an average of 1.5 time units to access each data item.

Access of blocks of other sizes and accesses that involve cells at different depths in the memory cause a degradation in performance.

depth number

Kluge's Memory Organization

Figure 3.7

3.5. Relative Performance

Some general remarks can be made about the dynamic memory configurations discussed so far. The Stone random access memory takes less time for random access takes longer for block accesses as compared to cyclic memories and the tour memories. These organizations allow a block access to proceed at the rate of one data access per time unit after the first few (one or two, as the case may be) accesses. Hence there is a trade-off made between random and block access times for these organizations. The Aho-Ullman memory, the Stone cyclic memory and the Wong-Tang memory organizations assume memory sizes that are one less than a power of 2. The Lenfant memory organization allows any size for the memory. Access times reported for the Lenfant memory are only empirical. Tour memory organizations have been shown feasible for certain sizes of memories, but it is not known whether tours exist for memories of all sizes. The Kluge memory is easier to implement on planar surfaces than the other organizations, but it does not have very good block access properties.

Table 3.3 summarizes the best access times that can be obtained using different memory organizations. Thus for an organization with good block access characteristics and with a fanout limitation of two the best coefficient that can be obtained for the logarithmic term in the access times is 1.893 for the worst case and 1.25 for the average case. With a fanout of 3 these coefficients have been brought down to 1.262 and 0.967 respectively.

<u>Organization</u>	<u>Cell</u> <u>Fanout</u>	<u>Random Access Time</u> <u>Worst</u>	<u>Average</u>	<u>Block Access Time</u> <u>Worst</u>	<u>Average</u>
Stone Random	2	$\hat{m}$	$\hat{m}-1$	$b \hat{m}$	$b \hat{m}$
Aho-Ullman	2	$1.893\hat{m}-2$	$1.25\hat{m}-1.333$	$2.5\hat{m}+b-4$	$1.25\hat{m}+b-1$
Stone Cyclic	2	$1.893\hat{m}$	$1.25\hat{m}$	$1.893\hat{m}+b-1$	$1.25\hat{m}+b-1$
2 Perm. Wong	2	$1.893\hat{m}-2$	$1.25\hat{m}-1.333$	$2.5\hat{m}+b-4$	$1.25\hat{m}+b-1$
3 Perm. Wong	3	$1.262\hat{m}-1$	$0.967\hat{m}-1.2$	$1.723\hat{m}+b-3$	$0.967\hat{m}+b-2.3$
Lenfant (emp)	2-6	--	$1.25\hat{m}$	--	$1.25\hat{m}+b-1$
Tour	2	$2\hat{m}-1$	$1.5\hat{m}-1$	$2\hat{m}+b-2$	$1.5\hat{m}+b-1$
Kluge	2	$2\hat{m}$	--	--	--

$$\hat{m} = \lceil \log N \rceil$$

N is the number of locations in the memory.

Block accesses are assumed to be for a block of b sequential data items.

-- : Not analyzed emp : empirical

Performance Comparison of Dynamic Memories
Table 3.3

In all of the organizations surveyed, except for Kluge's memory, the delay between two accesses is the same as the random access time for the latter access. Thus, delay expressions for these organizations are the same as access time expressions.

The following chapters present dynamic memory organizations that offer improvements in access times over all of the memories with two permutations and the Lenfant memory.

CHAPTER 4

THE DECK INTERCONNECTION NETWORK

4.1. Introduction

In the work surveyed in Chapter 3 and Chapter 4, most of the memory organizations require the fanout of every cell to be no more than 2. The best random access time is obtained by the Stone random access memory organization. For this organization the worst case random access time is $\log N$ and with a modified access algorithm [3] the average is $\log N - 1$. Requiring efficient block access of the memory organization raises the coefficient of $\log N$ in the expression for worst and average case random access times, as evident from the results for the Aho-Ullman memory, the Stone cyclic memory, the Wong-Tang two permutation memory and the tour memory.

Wong and Tang first relaxed the fanout restriction when they looked at the three permutation memory organization. For this organization, the coefficient of $\log N$ in the access time expressions is less than 1. Deletion of the fanout restriction brings a whole new class of dynamic memory organizations under scrutiny. Some of Lenfant's work, surveyed in chapter 4, has been in this direction. A dynamic memory organization that has an average cell fanout close to 2 is proposed in this chapter.

In all of the organizations surveyed earlier, the number of cells in the memory is equal to the number of data items in the

memory. This is not a necessary feature of a dynamic memory. In the organization described in this chapter extra cells will be used to facilitate access. The number of extra cells used will be small compared to the number of data items in the memory.

The major problem with the Stone random access memory is poor block access times. Partitioning the memory into modules and interleaving the addresses among these modules can help alleviate this problem. If implemented with silicon technology, all of these modules could be placed on a single chip appearing to the outside world as a single memory. The i/o port of each individual module will now be referred to as a distinguished location. Access techniques discussed in Chapter 2 are used to route data into distinguished locations of modules. Additional structures are needed to route data items accessed in each memory from their distinguished locations to the memory i/o port. This problem is trivial for most reasonably sized memories. In other cases it becomes necessary to temporarily store these accessed data items outside the memory modules, thus requiring additional cells.

The memory control mechanism consists of individual memory controllers, one for each module, obtaining directions from a global controller. Transfer time for control information from the global to the local controllers also contributes towards the access time.

Each module is organized as a Stone random access memory. The memory is partitioned into equal sized sets of cells and for each set the shuffle and shuffle exchange permutations are defined. The

overall organization is called the deck memory organization.

4.2. Structure of the Deck Memory

The deck memory can store N distinct data items with logical addresses in the range $[0, N-1]$. The memory consists of N cells that are partitioned into r sets of $\frac{N}{r}$ cells each. To simplify the control mechanism, N , $\frac{N}{r}$, and hence r are required to be powers of 2. Each set of $\frac{N}{r}$ cells is organized as a Stone random access memory and constitutes one module. The distinguished locations of the modules are connected to a "black box" via a bi-directional link. One additional cell, designated as the i/o port for the entire memory is also connected to the black box. An example of this organization with $N = 32$ and $r = 4$ is shown in Figure 4.1.

The function of the black box, which shall be referred to as the i/o propagation network, is to route incoming data to the appropriate memory module and to accept data from memory modules en route to the outside world. The i/o propagation network exploits the good random access behavior of the individual modules. The system, as a whole, performs block accesses efficiently.

Addresses are interleaved in the memory. If the modules are numbered from 0 through $r-1$, datum j resides in module $j \bmod r$. Associated with each datum within the module is a module address. Since each module consists of $\frac{N}{r}$ cells, the module address ranges from 0 to $\frac{N}{r} - 1$. Datum j in module $j \bmod r$ is defined to have module

$N = 31$

$r = 4$

Data Paths for a Deck Memory Organization

Figure 4.1

address $\left\lfloor \frac{j}{r} \right\rfloor$.

Control functions are divided among memory modules. A global controller is connected via another black box to the individual module controllers. The function of this black box, which shall be called the control propagation network, is to broadcast control information from the global controller to the module controllers. Access to datum j starts with the global controller being passed the address j . The controller recognizes that datum j resides in module $j \bmod r$ and makes the module address $\left\lfloor \frac{j}{r} \right\rfloor$ available to the local controller of the module. Control logic at each module consists primarily of the three registers described in Chapter 2. The access algorithm within each memory module is the same algorithm described in Figure 2.3. The major control paths in the deck memory are shown in Figure 4.2 for $N = 2^{11}$ and $r = 8$.

4.3. Access Times for the Deck Memory

Let t_c be the time required for control information to travel from the global controller to the module controller and t_a the time for data to pass between a distinguished location and the memory i/o port. Random access at a memory module involves the application of $\log \left(\frac{N}{r} \right)$ permutations within a module. The random read access time $t_{ra} = t_c + \log \left(\frac{N}{r} \right) + t_a$. For a write operation, the datum to be written may be routed from the i/o port to the appropriate distinguished location to arrive there exactly at the completion of

Broken lines represent Control Paths

Solid lines represent Data Paths

Figure 4.2

the access sequence within the module. This implies that the random write access time $t_{wa} = t_c + \log \left(\frac{N}{r} \right) - t_a$. This assumes that $t_a \leq t_c + \log \left(\frac{N}{r} \right)$.

Ignoring, for now, the specifics of the design of the propagation networks and the block access mechanism, it can be seen that access to r logically contiguous data items can proceed concurrently in the different memory modules. The r modules deliver these r data items to the i/o propagation network. The maximum data delivery rate of the memory is r data items per $\log \left(\frac{N}{r} \right)$ time. The memory is required to carry out block access at the rate of one data item per time unit after the access of the first datum in the block. This can be achieved if and only if $r \geq \log \left(\frac{N}{r} \right)$ and this serves as a defining equation for r . The i/o propagation network is designed so that access of a block of b contiguous data items takes $b - 1$ time units more than the random access time.

The i/o propagation network realizes an interconnection between the i/o port and the distinguished locations of the r modules. The number of data paths and the amount of logic required to realize the interconnection structure can be expected to rise at least linearly with r . Increasing r results only in a logarithmic decrease in the time $\log N - \log r$ which contributes to the random access time. This tradeoff between increasing cost and decreasing module access times means that as r is increased, a point of diminishing returns is quickly reached. In fact, as shall be seen in Chapter 5, for certain

deck memory designs random access time increases with r .

Minimization of r is hence sought.

Consider the functional $f()$ defined by

$$f(n) = \inf_p \{p \geq \log n - \log p \text{ and } p \text{ is a power of } 2\},$$

defined for $n > 1$. For example, $f(2^6) = 4$ since $4 = \log 2^6 - \log 4$.

Also $f(2^5) = 4$ since 4 is the smallest number p which is a power of 2

and satisfies the relation $p \geq \log 2^5 - \log p$. From the definition

of $f()$ it follows that for all integers $q > 0$, $f(n) = 2^q$ if and only

if $2^{2^{q-1}+q-1} < n \leq 2^{2^q+q}$. Restricting n to powers of 2 gives the

relation $2^{2^{q-1}+q} \leq n$ or $2^{q-1} + q \leq \log n$. Since $q > 0$, $2^{q-1} < \log n$

or $f(n) < 2 \log n$. For $q = 0$, $f(n) = 1$ and the only permissible

value for n is 2, which also satisfies the relation $f(n) < 2 \log n$.

Thus, $f(n) < 2 \log n$ for all $n > 1$, where n is restricted to be a power of 2.

The deck memory design seeks to minimize r while requiring r to

be a power of 2. This is done by setting $r = f(N)$. The random

access time for the deck memory is, therefore, $t_c + \log N - \log f(N)$

$\pm t_a$ (+ for read and - for write). Since $1 \leq f(N) < 2 \log N$,

$\log N \geq \log N - \log f(N) > \log N - \log \log N - 1$.

4.4. A Design for Small Deck Memories

Different designs may be proposed for the two propagation

networks in the basic structure of the deck. Figure 4.2 shows the

basic deck organization for $N = 2^{11}$ and $r = f(N) = 8$. The control

paths are represented by dotted lines and data paths are represented by solid lines. The i/o propagation network and the control propagation network in Figure 4.2 have to perform the simple functions of multiplexing and demultiplexing. Since the number r of memory modules is only 8, these functions can be performed by multiplexer/demultiplexers that can be easily constructed. Such a realization of the propagation networks result in a design of the deck memory called the simple deck.

Random access for the simple deck starts with the global controller being loaded with the address j of the datum to be accessed. The global controller chooses the module $j \bmod r$ that contains datum j and loads its local controller with the module address $\begin{bmatrix} j \\ r \end{bmatrix}$. This triggers a sequence of shuffles and shuffle exchanges within the module to bring datum j to the distinguished location. The local controller generates a signal to transfer the datum between the i/o port and the distinguished location. For a write operation, this transfer is done concurrently with the access permutation to end with the last permutation of the access sequence.

The global controller is designed to anticipate block access. Each time unit, it communicates with the next cyclically consecutive module controller. Module $(j+1) \bmod r$ accesses datum $j+1$ one time unit behind the access of datum j by module $j \bmod r$. This behavior is repeated for all subsequent data items in the block so that one datum reaches the i/o port every time unit after the first random access.

handle only one block access at a time, the delay is the same as the random access time.

4.5. Advantages of Partitioning

A major advantage of partitioning the Stone random access memory organization is that the memory can be made to have a small block access time. Two observations can be made with reference to possible semiconductor technology realizations of dynamic memories. Especially in the case of large memories, the partitioning of memory into several modules reduces the problem of crossing connections on the surface that houses the memory cells. Recent results by Hoey and Leiserson [14] indicate that the area for laying out a shuffle exchange graph on a planar surface is $O(\frac{N^2}{\log N})$ when $N = 2^m$ and m is a power of 2. Also, fabrication of circuitry on the device is simplified due to the replication of the modules constituting the deck.

CHAPTER 5

THE MULTI-LEVEL DECK MEMORY

5.1. Introduction

The simple deck memory organization was examined in Chapter 4. One criticism of the simple deck memory is the use of multiplexers and demultiplexers with the assumption that the time to transfer through these devices is negligible. Note that the number of memory modules for the simple deck $f(N)$, as defined in the previous chapter, is monotonically non-decreasing in N . The above assumption is not valid for large values of $f(N)$ for the storage technologies that motivate the dynamic memory model. Interconnections realizing the functions of the two propagation networks are presented in this chapter. For these interconnections, t_a and t_c are no longer independent of r , the number of memory partitions, and vary as $\log r$.

5.2. The Multi-level Deck Memory

The basic structure of the deck is summarized in Figure 5.1. This figure shows the memory partitions, the i/o port, the i/o propagation network, the local controllers, the global controller and the control propagation network.

The control propagation network for this design consists of $2r-1$ cells, each cell representing a node of a complete binary tree and is shown in Figure 5.2. The links represent uni-directional data paths

The Basic Structure of the Multi-Level Deck Memory

Figure 5.1

The Control Propagation Network Organization

Figure 5.2

going in the direction of the leaves from the root. Each cell is wide enough to contain control information being transferred from the global to the local controllers. The root of the tree is associated with the global controller and the leaves are associated with the module controllers. The depth of a cell in the tree is defined as the number of links that have to be traversed by control information to reach that cell, starting from the root node. Cells at depth i , $i \geq 1$, obtain control information from cells at depth $i-1$. The global controller loads control information into the root of the tree. The number of links to be traversed by the control information, traveling from the global controller to the module controllers, is $\log r$. The binary propagation network consists of $O(r)$ cells.

An interconnection network similar to the control propagation network can be used to interconnect the distinguished locations and the i/o port. All links in the binary tree will be bi-directional data paths. In this thesis an alternative to a binary tree structured i/o propagation network is examined. The resulting memory organization will be referred to as the multi-level deck memory. Since a multiplexer/demultiplexer pair cannot be used to route data between the i/o port and the distinguished locations the i/o propagation network must contain its own storage cells. Each of the r distinguished locations is connected to an additional memory cell as shown in Figure 5.1. These cells are labeled $D_0, D_1, \dots, D_{r-1}$, as shown in Figure 5.1. The distinguished location of module i is connected to cell D_i by a bi-directional data path.

When a block read access with starting address j is requested, groups of r consecutive data items in the block can be brought to these cells. The r cells must be interconnected to allow random read access to any of the r cells followed by access to adjacent cells. For a write access to the deck memory, the r cells $D_0, \dots, D_{r-1}$ are subject to a block write after which their contents are loaded into the distinguished locations of the partitions. To solve the dynamic memory access problem it becomes necessary to organize the cells $D_0, D_1, \dots, D_{r-1}$ as a memory with good random and block access properties. If $r < N$, the dynamic memory problem for N cells has been reduced to a smaller problem for a dynamic memory of r cells.

In this chapter the deck structure is chosen to recursively solve the dynamic memory problem. The r cells $D_0, D_1, \dots, D_{r-1}$ are organized to form a smaller deck structure. To distinguish the modules that contain these r cells from those constituting the original memory partitions, the r cells are said to form the L_2 -deck (for second level deck structure). The deck structure formed by the original memory partitions, is called the L_1 -deck (for first level deck structure). The reduction procedure may be recursively applied to result in a multi-level deck structure. Setting $r_0 = N$ and $r_1 = r$, the L_i -deck (for the i 'th level deck), consists of r_i modules of $\frac{r_{i-1}}{r_i}$ cells each, where r_i is a power of 2 and satisfies the inequality $r_i \geq \log \left(\frac{r_{i-1}}{r_i} \right)$. For $i \geq 2$ the r_{i-1} cells are connected

to the distinguished locations of the L_{i-1} -deck. i is called the level number of the L_i -deck. The multi-level deck structure is illustrated in Figure 5.3. The number of modules at a level decreases with increasing level number. The deck is extended in levels until, at the highest level, design of a simple deck becomes feasible. Assume that the k is the highest level number. In the extreme case, as will be assumed for the rest of this chapter, the deck is extended to level k where $r_k = 1$.

Given this structure the i/o and control network propagation times are no longer negligible. Local controllers are associated with each module at each level. Sufficient logic is built into these controllers so that the controllers need to know only the starting address and the block length to perform a block access. The r_i cells at depth $\log r_i$ of the control propagation network are associated with the r_i module controllers of the L_i -deck. Block access (starting with datum j) begins by the root cell of the control propagation network associated with the global controller being loaded with the address j and the block length. This information is propagated through the network and is available to all module controllers within $\log r_1$ time.

In the L_i -deck, a datum is accessed using a full access sequence of $\log \left(\frac{r_{i-1}}{r_i} \right)$ permutations. The datum so accessed need be available

The Complete Multi-level Deck Memory Organization

Figure 5.3

in the L_1 -deck only at the start of the access sequence in that deck. The access sequence for a datum consists of permutations generated by the access algorithm for modules at different levels as the datum passes through all of the decks. The access sequence for consecutive data items in a block access lag behind each other by one time unit at all levels. Specifically, the access sequence for datum $j+1$ lags behind the access sequence for datum j by one time unit.

For a block read access, the L_1 -deck fetches data items $j, j+1$, etc., into distinguished locations, one time unit apart and routes them to the i/o port through the i/o propagation network. For a block write access the L_1 -deck fetches the old data items $j, j+1$, etc., into distinguished locations while the i/o propagation network routes the new data items from the i/o port to the L_1 -deck where they are written in place of the old data items.

The control propagation network uses $2r_1 - 1$ cells. The number of cells used for control is minimized by minimizing r_1 . The total number of cells used in decks at levels higher than 1 is $r_1 + r_2 + \dots + r_k$. Since $r_{i+1} + \log r_{i+1} \geq \log r_i$, decreasing r_i will not force r_{i+1} to increase. Hence, minimization of the extra cells used in the higher level decks is achieved by minimizing r_1 , then minimizing r_2 , then r_3 , etc. Thus the number of extra cells used is minimized by setting $r_i = f(r_{i-1})$, $0 < i \leq k$. $f^i(n)$ represents the result of i

recursive applications of the functional $f()$ on n . Then $r_1 = f^1(N)$.

For example $r_2 = f^2(N) = f(f(N))$. Given N , a procedure has been established for the construction of the multi-level deck with k being the smallest number for which $f^k(N) = 1$. An example of the multi-level deck for $N = 2^{11}$ is illustrated in Figure 5.4.

5.3. Access Time Analysis

The time for a read access differs from the time for a write access in the multi-level deck memory. The deck is assumed to consist of k levels with $r_1 = f^1(N)$. Analysis is carried out for the worst case of $f^k(N) = 1$.

First, the read access time is calculated. t_c , the time for the propagation of control through $\log f(N)$ links, takes time $\log f(N)$.

Permutations in the L_1 -deck take $\log \left(\frac{f^{i-1}(N)}{f^1(N)} \right)$ time for each datum, 1

$\leq i \leq k$, where $f^0(N)$ is defined to be N . t_a , the propagation time

through the i/o propagation network, is $\sum_{i=2}^{k-1} \log \left(\frac{f^{i-1}}{f^1} \right) = \log f(N)$.

Propagation time through the i/o propagation network would also be $\log f(N)$ if it were structured as a complete binary tree. The random read access time $t_{ra} = t_c + \log N - \log f(N) + t_a = \log N + \log f(N)$.

For $N > 1$, $f(N) < 2 \log N$. Therefore $t_{ra} < \log N + \log (2 \log N) = \log N + \log (\log N) + 1$.

$$N = 2^{11}$$

$$r_1 = f(N) = 8, r_2 = f^2(N) = 2, r_3 = f^3(N) = 1$$

An Example of a Multi-level Deck Organization

Figure 5.4

Read accesses to data in a block after the access to the datum at the head of the block take one time unit for each data access. Thus, the read access time for a block of b data items is $t_{ra} + b - 1$.

The transmission of data from the i/o port to the L_1 -deck, for a write access, is done concurrently with the transmission of control information and the application of permutations in the L_1 -deck. The random write access time $t_{wa} = t_c + \log N - \log r - t_a = \log N - \log f(N)$.

The access time for a block write of b consecutive words is $t_{wa} + b - 1$.

The multi-level deck proposed in this chapter has an access time of $O(\log N)$ and uses an additional $O(\log N)$ cells. The average fanout per cell does not increase significantly beyond 2.

5.4. Heavily Loaded Multi-level Deck

5.4.1. Introduction

The memory is often a bottleneck in performance of computer systems. In many systems, queues build up for memory access and the queues seldom empty. A memory is said to be heavily loaded if there is always at least one access request that is queued for memory service that has not received any service. All accesses are assumed to be block accesses. Under conditions of heavy load, an important parameter of performance is the average delay, where delay is defined as the time between block accesses during which there are no data

transfers taking place between the memory and the outside world.

The access mechanism described in the previous section will be summarized here. In Figure 5.3 the L_1 -deck, containing all the current data, is called as the data level.

The read access mechanism is summarized by the following steps.

- 1) Pass control information through the control propagation network in $\log r$ time (using r for r_1).
- 2) Sequentially trigger access permutation sequences in the modules of the data level. A datum is brought to a distinguished location in $\log \left(\frac{N}{r}\right)$ time.
- 3) Set off the operation of the i/o propagation network to obtain a steady stream of data arriving from the distinguished locations at the i/o port. Passage of a datum through the i/o network takes $\log r$ time.

The total random read access time is $\log N + \log r$.

The following steps summarize the write access strategy.

- 1) Pass control information through the control propagation network in $\log r$ time.
- 2) Bring the old contents of logical addresses (that are to be written into) to the distinguished locations. This takes $\log \left(\frac{N}{r}\right)$ time.
- 3) Overlap steps 1 and 2 with transmission of data from the i/o port to the data level. Transmission through the i/o propagation network takes $\log r$ time. However, transmission of

data through the i/o propagation network is overlapped to end at the same time as the first two steps. No extra time is contributed towards the access.

The total random write access time is $\log N - \log r$ time units.

In the rest of this section, the memory is assumed to be heavily loaded. The memory organization will be modified to service more than one request at a time. This is done through the overlapping of some of the steps of the access operations for consecutive requests. The amount of overlap depends on the nature of the overlapping requests, i.e., whether they are read accesses, or write accesses, and the locations being accessed. Assume that the set of requests consists of independent read and write block access requests with random block lengths. Also assume that starting block addresses are uniformly distributed over the r memory modules of the data network. The delays between the block accesses will be calculated, assuming that the first block access ends with access to datum v and the second block access starts with datum j .

5.4.2. Read-Read Delay

The time interval between the arrivals of datum v and datum j at the memory i/o port is the delay between the two accesses. The passage of control information for access of datum j can be done concurrently with the access of v .

If datum j and datum v belong to the same memory module of the data level then the permutations for the access of v have to be

completed before the access permutations for j can begin. For a period of $\log(\frac{N}{r})$, no output will be available at the memory i/o port. This results in the worst case delay of $\log(\frac{N}{r})$ time units.

If datum v and datum j are in different modules then the latter access permutations for v can be overlapped with the initial access permutations for j . To calculate the average delay assume, without loss of generality, that datum j resides in module 0. With probability $\frac{1}{r}$ datum v resides in module i , $0 \leq i < r$. The delay is expressed as a function of the module in which datum v resides (see Table 5.1). The average delay evaluates to

$$\frac{[\log(\frac{N}{r})][\log(\frac{N}{r})-1]}{2r} + 1 \leq \frac{\log(\frac{N}{r})-1}{2} + 1, \text{ since}$$

$$r \geq \log(\frac{N}{r}).$$

The control mechanism is more complex when consecutive accesses are overlapped. It becomes necessary to provide buffers for the controller for each module, to queue control information. This requires the use of a large number of extra cells. Queues of buffers can be eliminated if the control information were to be passed from one level of deck to another along with the passage of the data which is being accessed. This thesis does not consider the details of such an organization.

<u>Module Containing Datum v</u>	<u>Delay</u>
0	$\log \left(\frac{N}{r} \right)$
1	$\log \left(\frac{N}{r} \right) - 1$
2	$\log \left(\frac{N}{r} \right) - 2$
.	..
.	..
$\log \left(\frac{N}{r} \right) - 1$	1
$\log \left(\frac{N}{r} \right)$	1
$\log \left(\frac{N}{r} \right) + 1$	1
.	..
.	..
r-1	1

Table 5.1
Delays As Function Of Module Containing Datum v
For Read Followed By Read.

5.4.3. Read-Write Delay

The read-write delay is the difference between the time datum v is brought to the i/o port and the time datum j is written into the i/o port.

The worst case occurs when datum j and datum v belong to the same module in the data level. After loading datum v into the i/o network, the module requires $\log \left(\frac{N}{r} \right)$ time units before it is ready to accept datum j at its distinguished location. During this interval, datum v moves to the i/o port, there is a delay between the two accesses and then datum j is routed from the i/o port to the module.

Passage through the i/o propagation network takes $\log r$ time units, in either direction. The worst case delay is $\log \left(\frac{N}{r}\right) - 2 \log r$.

This assumes that $\log \left(\frac{N}{r}\right) \geq 2 \log r$ or $N \geq r^3$, which is true for $N \geq 2^{11}$.

To calculate the average delay, assume that datum j resides in module 0. The delay, evaluated in Table 5.2, is a function of the module in which v resides. The average delay is $\frac{[\theta][\theta-1]}{2r} + 1$, where $\theta = \log \left(\frac{N}{r}\right) - 2 \log r$.

<u>Module Containing Datum v</u>	<u>Delay</u>
0	$\log \left(\frac{N}{r}\right) - 2 \log r$
1	$\log \left(\frac{N}{r}\right) - 2 \log r - 1$
2	$\log \left(\frac{N}{r}\right) - 2 \log r - 2$
.	..
.	..
$\log \left(\frac{N}{r}\right) - 2 \log r - 1$	1
$\log \left(\frac{N}{r}\right) - 2 \log r$	1
$\log \left(\frac{N}{r}\right) - 2 \log r + 1$	1
.	..
.	..
r-1	1

Table 5.2
Delays As Function Of Module Containing Datum v
For Read Followed By Write.

5.4.4. Write-Write Delay

The write-write delay is the difference between the times at which data items v and j are written into the memory i/o port.

The worst case occurs when datum j and datum v belong to the same memory module in the data network. Permutations for the access of datum j are applied only after the access sequence for datum v is completed in the module. This causes a delay of $\log(\frac{N}{r})$ time units.

The delay is evaluated as a function of the module containing datum v in Table 5.3, assuming that datum j resides in module 0. The

average delay is $\frac{[\log(\frac{N}{r})][\log(\frac{N}{r})-1]}{2r} + 1$.

Module Containing Datum v	Delay
0	$\log(\frac{N}{r})$
1	$\log(\frac{N}{r}) - 1$
.	..
$\log(\frac{N}{r}) - 1$	1
$\log(\frac{N}{r})$	1
.	..
.	..
$r-1$	1

Table 5.3
Delays As Function Of Module Containing Datum v
For Write Followed By Write.

5.4.5. Write-Read Delay

The write-read delay is the delay between a write access and the following read access. It is the interval between the writing of datum v into the i/o port and the reading of datum j from the i/o port.

The worst case delay is experienced if datum j and datum v reside in the same module in the data network. The transmission times of datum j and datum v through the i/o network is added to the time required to apply the access permutations for j , to result in a worst case delay of $\log \left(\frac{N}{r} \right) + 2 \log r$.

If datum j and datum v reside in different modules, then parts of the two accesses may be overlapped, requiring the i/o propagation network to handle inputs and outputs simultaneously. This involves transferring data in two directions concurrently. A single i/o propagation network is inadequate. The solution proposed for the multi-level deck is to replicate the i/o propagation network, with one network routing only inputs and the other routing only outputs. Note that this modification is necessary only to reduce delays.

Without loss of generality, datum j is assumed to be in module 0. The delay between the accesses (see Table 5.4 and Table 5.5) is expressed as a function of the module in which datum v resides. Two cases arise based on the relative values of r and $\log N + \log r$. The delays for these two cases are separately calculated in Table 5.4 and Table 5.5. Let $\beta = \log N + \log r$. In the first case ($r \geq \beta$) the

<u>Module Containing Datum v</u>	<u>Delay</u>
0	$\log \left(\frac{N}{r} \right) + 2 \log r$
1	$\log \left(\frac{N}{r} \right) + 2 \log r - 1$
2	$\log \left(\frac{N}{r} \right) + 2 \log r - 2$
.	..
.	..
$\log \left(\frac{N}{r} \right) + 2 \log r - 1$	1
$\log \left(\frac{N}{r} \right) + 2 \log r$	1
.	..
.	..
r-1	1

Table 5.4
Case 1 : $r \geq \log N + \log r$
Delays As Function Of Module Containing Datum v
For Write Followed By Read.

<u>Module Containing Datum v</u>	<u>Delay</u>
0	$\log \left(\frac{N}{r} \right) + 2 \log r$
1	$\log \left(\frac{N}{r} \right) + 2 \log r - 1$
2	$\log \left(\frac{N}{r} \right) + 2 \log r - 2$
.	..
.	..
r-1	$\log \left(\frac{N}{r} \right) + 2 \log r - r + 1$

Table 5.5
Case 2 : $r < \log N + \log r$
Delays As Function Of Module Containing Datum v
For Write Followed By Read.

average delay is $\frac{\beta[\beta-1]}{2r} + 1$. In the other case ($r < \beta$) the average

delay is $\frac{\beta[\beta+1]}{2r} - \frac{[\beta-r][\beta-r+1]}{2r}$.

5.4.6. Expected Delay For The Deck Memory

The worst and average delays for the multi-level deck are summarized in Table 5.6. Accesses are assumed to be independent of each other. p is the probability of an access being a read access. $q = 1 - p$ is the probability of the access being a write access. The average delay between consecutive accesses is given by $\bar{t}_d = p^2 \bar{t}_{rr} + pq \bar{t}_{rw} + q^2 \bar{t}_{ww} + pq \bar{t}_{wr}$, where

$\bar{t}_{rr}$ = the average delay for read followed by read
 $\bar{t}_{rw}$ = ,, ,, ,, ,, read ,, ,, write
 $\bar{t}_{ww}$ = ,, ,, ,, ,, write ,, ,, write
 $\bar{t}_{wr}$ = ,, ,, ,, ,, write ,, ,, read.

$\bar{t}_d$ is evaluated separately for the two cases described in the

<u>Access Description</u>	<u>Worst Case Delay</u>	<u>Average Case Delay</u>
Read-Read	$\log \left(\frac{N}{r} \right) = \alpha$	$\frac{\alpha[\alpha-1]}{2r} + 1$
Read-Write	$\log \left(\frac{N}{r} \right) - 2 \log r = \theta$	$\frac{\theta[\theta-1]}{2r} + 1$
Write-Write	$\log \left(\frac{N}{r} \right) = \alpha$	$\frac{\alpha[\alpha-1]}{2r} + 1$
Write-Read	$\log \left(\frac{N}{r} \right) + 2 \log r = \beta$	Case 1 : $r \geq \beta$ $\frac{\beta[\beta-1]}{2r} + 1$ Case 2 : $r < \beta$ $\frac{\beta[\beta+1]}{2r} - \frac{[\beta-r][\beta-r+1]}{2r}$

Table 5.6
Summary Of Delays In The Deck Memory

previous section.

Case 1 : $r \geq \log N + \log r$.

$$\begin{aligned}
 \bar{t}_d &= p^2 \left[\frac{\log(\frac{N}{r})[\log(\frac{N}{r})-1]}{2r} + 1 \right] \\
 &\quad + pq \left[\frac{[\log(\frac{N}{r})-2\log r][\log(\frac{N}{r})-2\log r-1]}{2r} + 1 \right] \\
 &\quad + q^2 \left[\frac{\log(\frac{N}{r})[\log(\frac{N}{r})-1]}{2r} + 1 \right] \\
 &\quad + pq \left[\frac{[\log(\frac{N}{r})+2\log r][\log(\frac{N}{r})+2\log r-1]}{2r} + 1 \right] \\
 &= 1 + \frac{\log(\frac{N}{r})[\log(\frac{N}{r})-1]}{2r} + \frac{2pq}{2r} [4 \log^2 r] \\
 &\leq \frac{1}{2} \log(\frac{N}{r}) + \frac{5}{8} + 1,
 \end{aligned}$$

since, $r \geq \log(\frac{N}{r})$, $(pq)_{\max} = \frac{1}{4}$ and $(\frac{\log^2 r}{r})_{\max} = \frac{9}{8}$ for r being a power of 2.

Case 2 : $r < \log N + \log r$

$$\begin{aligned}
 \bar{t}_d &= p^2 \left[\frac{\log(\frac{N}{r})[\log(\frac{N}{r})-1]}{2r} + 1 \right] \\
 &\quad + pq \left[\frac{[\log(\frac{N}{r})-2\log r][\log(\frac{N}{r})-2\log r-1]}{2r} + 1 \right]
 \end{aligned}$$

$$\begin{aligned}
& + q^2 \left[\frac{[\log(\frac{N}{r})][\log(\frac{N}{r})-1]}{2r} + 1 \right] \\
& + pq \left[\frac{[\log(\frac{N}{r})+2\log r][\log(\frac{N}{r})+2\log r+1]}{2r} + 1 \right] \\
& - pq \left[\frac{[\log(\frac{N}{r})+2\log r-r][\log(\frac{N}{r})+2\log r-r+1]}{2r} \right] - pq \\
& \leq \frac{1}{2} \log(\frac{N}{r}) + 1 + 4pq \left(\frac{\log^2 r}{r} + \frac{\log r}{2r} \right) - \frac{1}{2} + pq \\
& \leq \frac{\log(\frac{N}{r})}{2} + \frac{17}{16} + 1,
\end{aligned}$$

since, $r \geq \log(\frac{N}{r})$, $(pq)_{\max} = \frac{1}{4}$, $(\frac{\log^2 r}{r} + \frac{\log r}{2r})_{\max} = \frac{21}{16}$ when r is a power of 2 and $r < \log N + \log r$.

$r = f(N)$ for the multi-level deck. Therefore

$$\frac{1}{2} (\log N - \log f(N)) + \frac{17}{16} + 1$$

is an upper bound for the expected delay for the multi-level deck memory.

In many computer applications read requests occur more frequently than write requests. Setting $p = 0.75$ and $q = 0.25$ assumes that reads are three times as frequent as writes. For these values of p and q $\bar{t}_d \leq \frac{1}{2} \log \frac{N}{r} + \frac{11}{32} + 1$ for the first case discussed above and $\bar{t}_d \leq \frac{1}{2} \log \frac{N}{r} + \frac{43}{64} + 1$ for the other.

CHAPTER 6

CONCLUSIONS

6.1. Performance Comparison

In Chapter 3, the performances of memory organizations proposed in the literature were presented. The performance measures of interest were the worst and average case random access times and block access times. The simple deck and multi-level deck will be compared with the memory organizations surveyed earlier, using these measures plus the worst and average case delays.

For all organizations except the deck and Kluge's organization, permutations for two consecutive but independent accesses can not be overlapped. For these organizations the delay expressions are the same as the random access time expressions.

As described in Chapter 4, access permutations of independent consecutive block accesses to the simple deck are not done concurrently. Sufficient intelligence may be built into the controllers so that only the starting addresses and the length of a block requested need be broadcast from the global controller for complete service of a block access request. This allows the potential overlap of access permutations of consecutive independent block accesses. Consider two consecutive block accesses to such a memory. Assume that the first block access ends with access to datum v and that the subsequent block access starts with the access to

datum j . Some of the access permutations for these two data items can be applied concurrently if the data items reside in different modules. The worst case delay between these two accesses occurs when data items v and j reside in the same memory module. This delay is $\log N - \log f(N)$. Assume that starting block addresses are independently and uniformly distributed over the r memory modules. Then the average delay for the simple deck is

$$\frac{\log(\frac{N}{f(N)})[\log(\frac{N}{f(N)}) - 1]}{2f(N)} + 1 \leq 1/2 \log(\frac{N}{f(N)}) + 1/2.$$

Table 6.1 provides a concise comparison of the optimum performances of the Stone random memory, the Aho-Ullman memory, the Stone cyclic memory, the Wong-Tang memories, the Lenfant memory, the Tour memory, the Kluge memory, the simple deck memory and the multi-level deck memory (as a function of N , the number of distinct data items that can be stored in the memory). In the case of the cyclic memories the same organization does not show the best behavior with respect to all performance parameters. The performance figures quoted in a row of Table 6.1 may not refer to the same member of the class of organizations. Other parameters tabulated are the average fanout of a cell and the number of cells used to implement the memory.

Comparison of the memory organizations in Table 6.1 indicates the advantages offered by the deck memory over other memory organizations. The average case random access and block access times are smaller for the Wong-Tang three permutation memory. However, the

Memory	Fan out	Num. Cells	Random Access Time Worst	Avg.	Block Access Time Worst	Avg.	Worst	Delay Avg.
Stone Random	2	N	x	x-1	--	--	--	--
Aho-Ullman	2	N	1.89x-2	1.25x-1.33	2.5x+b-4	1.25x+b-1	1.89x-2	1.25x-1.33
Stone Cyclic	2	N	1.89x	1.25x	1.89x+b-1	1.25x	1.89x	1.25x
Wong 2 Per.	2	N	1.89x-2	1.25x-1.33	2.5x+b-4	1.25x+b-1	1.89x-2	1.25x-1.33
Wong 3 Per.	3	N	1.26x-1	0.96x-1.2	1.72x+b-3	0.96x+b+0.97	1.26x-1	0.96x-1.2
Lenfant (emp)	2-6	N	--	1.25x	--	1.25x+b-1	--	1.25x
Tour	2	N	2x-1	1.5x-1	2x+b-2	1.5x+b-1	2x-1	1.5x-1
Kluge	2-	N	2x	--	--	--	2x	--
Simple Deck~	2	N+1	x-y	x-y	x-y+b-1	x-y+b-1	x-y	$\leq \frac{1}{2} (x-y) + \frac{1}{2}$
Multi-deck (+ for read - for write)	2+z	N+	$\overline{x+y}$	$\overline{x+y}$	$\overline{x+y+b-1}$	$\overline{x+y+b-1}$	x+y	$\leq \frac{1}{2} (x+y) + \frac{33}{16}$

emp : empirical -- : not analyzed ~ : t_c and t_a are ignored

$$x = \lceil \log_2 N \rceil \quad y = \log_2(f(N)) \quad z : O(\log N)$$

Performance Comparison of Dynamic Memory Organisations

Table 6.1

deck outperforms the Wong-Tang organization with respect to worst case random access and block access times and worst and average case delays. On the basis of the delay expressions, the deck seems best suited for operation under heavily loaded conditions.

Note that the Wong-Tang three permutation organization has a cell fanout of 3. By going to a cell fanout of $3+$ from $2+$, for the deck memory organization, access times can be reduced by a factor of $\frac{1}{\log 3}$. The number of extra cells used by the deck would also be reduced proportionally.

Using the deck organization results in improved performance for the dynamic memory when compared with other dynamic memory organizations. This improvement is obtained at a cost - the complexity of the control mechanism has increased considerably. The interconnection structure of the memory also may have increased in complexity, making implementation difficult. The average cell fanout has increased to slightly above 2. $O(\log N)$ extra cells are used, adding to the cost of the memory. Whether or not the increased performance justifies the additional complexity is at this point an unresolved issue.

6.2. Extensions and Open Problems

The deck has been proposed for $N = 2^m$. This facilitates the realization of the control mechanism. Generalizations of the Stone random access memories made by Morris, et. al. [13], indicate that the deck memory organization can be generalized to any arbitrary

value of N . One extension to this research would be to investigate the effect of such a generalization on access times and delays.

It is conjectured that the multi-level deck, or a small modification thereof, could provide efficient access when the unit of access is a block of r data items, where r is the number of modules in the L_1 -deck. This may be of particular value for paged memories where the page size is r words.

One aspect of the dynamic memory problem that has drawn little attention so far is the space complexity of the organization. The space complexity is a measure of the space required to realize the memory cells and their interconnection and control structures. For a VLSI implementation, one would calculate the amount of space required to layout the circuitry for the memory on the surface of a chip. Modules at the lower levels of the multi-level deck may be numbered so that elements belonging to higher level decks are adjacent to each other. The problem of organizing and arranging a dynamic memory for efficient VLSI implementation is a potentially important one and merits further investigation.

A more pragmatic approach may be taken to tackle the dynamic memory access problem. One could start with easily implementable permutations for the memory as given and then search for efficient access schemes using these permutations. This suggests the study of specific technologies and the need to identify the primitive data movement capabilities of these storage technologies.

A more ambitious and difficult problem is to develop a theory that relates the number of cells in the memory, the number of data items, the fanout of cells, the random access times, the block access times, and the delays. Some of the trade-offs that can be made among these factors are illustrated in Table 6.1. A quantitative evaluation of the potential trade-offs is desirable.

6.3. Summary

The dynamic memory access problem has been defined in this thesis. Several solutions proposed in the literature have been surveyed. One such solution called the Stone random access memory formed the basis of a partitioned interleaved memory called the deck. The deck is simple and easily implemented with multiplexers for memories of reasonable sizes. The structure of a very large deck memory has also been discussed. The performance of the deck memory appears to be better than of most of the other proposed organizations.

APPENDIX A

The Aho-Ullman Memory

Aho and Ullman [7] consider a dynamic memory of size $N = r^m - 1$, where r and m are integers. The memory locations are numbered from 0 to $N-1$, with physical location 0 as the i/o port. Two permutations d_s^r and d_c are defined.

$$d_s^r(j) = rj \bmod N$$

$$d_c(j) = (j - 1) \bmod N, \text{ where } 0 \leq j < N.$$

If j is represented as a number base r , then $d_s^r(j)$ is simply the base r representation of j rotated left one digit.

The initial memory map is given by $s(1) = 1$. Aho and Ullman show that the memory map always has the form $s(j) = r^p j + q \bmod N$, for integers p and q , where $0 \leq p < m$ and $0 \leq q < N$. In terms of parameters p and q

$$d_s^r \text{ maps } \begin{cases} p \text{ into } p+1 \bmod m \\ q \text{ into } rq \bmod N \end{cases}$$

and

$$d_c \text{ maps } \begin{cases} p \text{ into } p \\ q \text{ into } (q-1) \bmod N. \end{cases}$$

The memory control mechanism keeps track of p and q . To access datum j , the control mechanism generates a sequence of d_s^r 's and d_c 's.

If $s(j)$, the physical addresses of datum j , is considered as a base r number then the permutation d_c is used to zero the least significant digit of $s(j)$ if necessary and then d_s^r is used to rotate the representation by one digit. This procedure is continued until datum j reaches the i/o port. The access algorithm is described in Figure A.1.

The worst case random access time, as determined by Aho and Ullman, is $r \log_r(N+1) - 2$; i.e., $\frac{r}{\log r} \log(N+1) - 2$. $\frac{r}{\log r}$ is evaluated for different values of r in Table A.1. The worst case random access time is minimized to $1.893 \log(N+1)$ when $r = 3$.

When datum j is brought to the i/o port (location 0) the following is true of the memory map:

$$s(j) = r^p j + q = 0.$$

Now, datum $j+1$ is in location $s(j+1) = r^p(j+1) + q = r^p$, where all arithmetic is modulo N . To access datum $j+1$ the access algorithm generates $m-p \bmod m$ d_s^r 's followed by a d_c . The number of

r	$\frac{r}{\log r}$
2	2.000
3	1.893
4	2.000
5	2.153
6	2.321
7	2.493
8	2.667

Table A.1

$$\frac{r}{\log r} \text{ vs } r$$

Input: p, q (memory map) and j (the datum to be accessed).

Output: Access sequence, updated p and q (memory map).

```

        locj ← rpj + q
loop:   until lsd(locj) = 0 do
        { dc; locj ← locj-1; q ← q-1 }

        if locj ≠ 0 do
        { ds; locj ← r*locj; q ← r*q

          p ← (p+1) mod m; goto loop }

```

lsd : least significant digit

r*q and r*locj represent base r left cyclic shifts
of q and locj respectively

Access Algorithm for the Aho-Ullman Memory

Figure A.1

permutations so generated is between 1 and m . After this access, the memory map is of the form $s(i) = r^0 i + q$, where $s(j+1) = j + 1 + q' = 0$. Hence $q' = -(j+1)$, $p = 0$. Subsequent d_c permutations bring consecutive data items into the i/o port, at unit time intervals.

The worst case block access time is $r \log_r(N+1) - 2 + \log_r(N+1) + b - 2$, which can be rewritten as $\frac{(r+1)}{\log r} \log(N+1) + b - 4$. $\frac{(r+1)}{\log r}$ is calculated for different values of r in Table A.2. $r = 4$ minimizes the worst case time for the access of a block of b data items to $2.5 \log(N+1) + b - 4$ time units.

Aho and Ullman have analyzed the average time for random access, t_{av} . The expression for t_{av} is derived below as a special case of the access time analysis carried out by Wong and Tang for their two permutation memory. It is assumed that random accesses to logical locations are equiprobable.

r	$\frac{(r+1)}{\log r}$
2	3.000
3	2.524
4	2.500
5	2.584
6	2.708
7	2.850
8	3.000

Table A.2

$\frac{(r+1)}{\log r}$ vs r

$$\text{Then } t_{av} = \frac{(r^m - 2)(r - 1)m}{2(r^m - 1)} + m - \left[1 + \frac{r^m - r}{(r - 1)(r^m - 1)}\right],$$

where $m = \log_r(N+1)$. For large values of m , t_{av} is approximated by $\frac{(r+1)}{2 \log r} \log(N+1) - \frac{r}{(r-1)}$. Table A.3 shows how $\frac{(r+1)}{2 \log r}$ varies with r . $r = 4$ minimizes the average random access time to $1.25 \log(N+1) - 1.333$. Access of datum $j+1$, immediately thereafter takes, in the worst case, m time units. The average block access time is between $t_{av} + b - 1$ and $t_{av} + m + b - 2$.

Assume that only block accesses, with block size ≥ 2 , are requested of the memory. Assume that if the memory is not accessing any datum then d_c permutations are applied to the memory. This assumption need not be made if the memory is never idle, as would be the case for heavily loaded memories. The assumptions made ensure that, at the beginning of every block access, the memory map is of the form $s(i) = i \oplus q$, for some q , $0 \leq q < N$.

r	$\frac{(r+1)}{2 \log r}$
2	1.500
3	1.262
4	1.250
5	1.292
6	1.352
7	1.425
8	1.500

Table A.3

$$\frac{(r+1)}{2 \log r} \text{ vs } r$$

Assume that the starting address of the blocks are uniformly distributed over the range of addresses $(0, N-1)$. Therefore the physical address of the head of the block to be accessed is uniformly distributed over the same address range. Without loss of generality, assume that, at the beginning of the block access request, the memory map is of the form $s(j) = j$, with $p = 0$ and $q = 0$. The average access time for the first datum in the block is the average random access time, as calculated above.

Next, the average access time for the second datum in the block is computed. Consider the sequence of d_m^r and d_c permutations, generated by the access algorithm to access datum k , the head of the block. Let p be the number of d_s^r permutations generated. Datum k initially resides in physical location k . At the conclusion of the access of datum k , the memory map would be of the form $s(j) = r^p j + q$ with $s(k) = 0$. Access to datum $k+1$ takes time $(m-p \bmod m) + 1$.

The range of k , $(0, r^m - 2)$, can be divided into several classes, based on the r -ary representation of the address. For all k in a class, the access time for datum $k+1$ is the same. Consider the base r representation of k , i.e., $b_{m-1}b_{m-2}b_{m-3}\dots b_1b_0$. Table A.4 describes the general form of addresses in each class. Here ' n ' stands for any non-zero base r digit and '*' stands for any base r digit. Table A.4 indicates the number of d_s^r permutations applied to the memory for the access of datum k , and the number of steps required for the access of datum $k+1$, for each class. The number of

distinct addresses in each class is also tabulated.

The average access time for datum $k+1$ is the sum of the access times computed for the classes weighted by the probability that address k belongs to that class. Since all addresses in the given address range are equiprobable, the probability of the address k belonging to a particular class is proportional to the number of addresses in that class. The average access time for datum $k+1$ is

$$t_s =$$

$$\frac{1 + (r-1) + m(r-1)r + (m-1)(r-1)r^2 + \dots + (m-(m-2))(r-1)r^{m-1} - 2}{2^m - 1}$$

$$= \frac{1 + (r-1) - 2 + m(r-1) \sum_{i=1}^{m-1} r^i - (r-1) \sum_{i=1}^{m-2} i r^{i+1}}{r^m - 1}$$

Address Form of k	Num. of d_s^r , permutations to access k	Num. of steps to access $k+1$	Num. of members in class
000 .. 00	0	1	1
000 .. 0n	0	1	$(r-1)$
n00 .. 0*	1	m	$(r-1)r$
n0 .. 0	2	$m-1$	$(r-1)r^2$
**n .. 0*	3	$m-2$	$(r-1)r^3$
.. .. .	.	...	
.. .. .	.	...	
.. .. .	.	...	
*** ** n*	$m-1$	2	$(r-1)r^{m-1} - 1$

Table A.4

Access Time for second datum in a block access

$$= \frac{(r-2) + m(r^m-r) - \frac{r^2}{r-1} [(m-2)r^{m-1} - (m-1)r^{m-2} + 1]}{r^m-1}.$$

For large values of m , t_s is approximated by $\frac{2r-1}{r-1}$.

The average block access time is

$$\frac{(r^m-2)(r-1)m}{2(r^m-1)} + m - \left[1 + \frac{(r^m-r)}{(r-1)(r^m-1)}\right] + \frac{(r-2)}{(r^m-1)} + \frac{m(r^m-r)}{(r^m-1)} - \frac{r^2[(m-2)r^{m-1} - (m-1)r^{m-2} + 1]}{(r-1)(r^m-1)} + b - 2.$$

For large values of m this is approximated by $\frac{r+1}{2 \log r} \log (N+1) - 1 +$

b .

$\frac{r+1}{2 \log r}$ has been computed in Table A.3. $r=4$ minimizes the block

access time to $1.25 \log (N+1) + b - 4$.

APPENDIX B

The Stone Cyclic Memory

Stone [8] considers a slight modification of the Aho and Ullman memory. The memory consists of $N = r^m - 1$ locations numbered from 1 through N , rather than 0 through $N-1$. The memory stores N data items. The memory locations are considered as one set. The two memory permutations defined for the memory are

$$d_s^r(i) = ri \bmod (N+1) \text{ for } 1 \leq i < N$$

$$d_s^r(N) = N$$

$$d_c(i) = i \oplus (-1) \text{ for } 0 < i \leq N,$$

where $\oplus$ denotes diminished radix complement addition [15]. The d_c permutation moves the datum in location 1 to location N because in diminished radix complement arithmetic

$$\begin{array}{rcl} 00 \dots 01 & & 1 \\ rr \dots r(r-1) & \oplus & (-1) \\ \hline rr \dots rr & & N. \end{array}$$

The integer $rr \dots rr$ is negative zero in the diminished radix complement notation and is the representation of N in unsigned base r representation. In this memory organization location 1 is the i/o port.

Stone assumes that the data items are arranged sequentially in memory after every access. In fact if data items are arranged sequentially before an access, then the access sequence generated leaves the memory in a cyclic shift of the initial state. Before every access, the memory map can be written as $s(i) = i \oplus q$. Let $t = i \oplus q$ and let $(b_{m-1}, b_{m-2}, \dots, b_1, b_0)$ be the base r representation of t . The access algorithm is described in Figure B.1.

At the end of the random access, data items are arranged sequentially in memory. Access to consecutive data items requires one application of permutation d_c per item. The worst case time for random access in this memory is $r \log_r(N+1) = \frac{r}{\log r} \log(N+1)$. This can be compared with the worst case random access time for the Aho and Ullman organization which is $\frac{r}{\log r} \log(N+1) - 2$. $\frac{r}{\log r}$ has been evaluated for several values of r in Table A.1. $r = 3$ gives the minimum worst case random access time of $1.893 \log(N+1)$.

The average random access time can be calculated by assuming that the random accesses are uniformly and independently distributed over all addresses in the address range of the memory. Based on this assumption, the average random access time is $\log_r(N+1) +$

$\frac{r-1}{2} \frac{r^m}{r^m-1} \log_r(N+1)$. The additional multiplicative term $\frac{r^m}{r^m-1}$ appears

because 0 does not belong to the address range. The average access time can be approximated for large values of m by $(\frac{r+1}{2}) \log_r(N+1)$,

rewritten as $\frac{r+1}{2 \log_2 r} \log(N+1)$. This can be compared with the

Input: j (datum to be accessed) and i (datum in i/o port);

Output: Access Sequence

$$\text{Let } j - i = \sum_{t=0}^{m-1} b_t.$$

```

for k = m-1 step -1 until 0 do
    begin
        Apply ds;
        Apply dc bk times;
    end
end

```

Access Algorithm For The Stone Cyclic Memory

Figure B.1

average access time for the Aho and Ullman memory which is

$(\frac{r+1}{2})\log_r(N+1) - \frac{r}{r-1}$. Computation of $\frac{r+1}{2 \log r}$ in Table A.3 indicates that for $r = 4$, the memory has a minimum average random access time of $1.25 \log(N+1)$.

The Aho and Ullman memory has a slight edge over the Stone memory in terms of both worst case random access time and average case random access times. A major virtue of the Stone memory lies in a simplified access algorithm which replaces m -bit additions by bit comparisons and cyclic shifts. The Aho and Ullman access algorithm has potential delays due to carry propagation in every iteration.

A block access proceeds at the rate of one datum per time unit after first random access of the datum at the head of the block. The worst case time to access a block of b data items is given by $r \log_r(N+1) + b - 1$ and the average case access time is $\log_r(N+1) + \frac{r-1}{2} \frac{r^m}{r^m-1} \log_r(N+1) + b - 1$ which, for large values of m , can be approximated by $(\frac{r+1}{2})\log_r(N+1) + b - 1$. This is the same as the average block access time for the Aho and Ullman memory. $r = 3$ gives the minimum worst case block access time of $1.893 \log(N+1) + b - 1$ and $r = 4$ gives the minimum average block access time of $1.25 \log(N+1) + b - 1$.

APPENDIX C

The Two Permutation Wong-Tang Memory

Wong and Tang have proposed a generalization [10] of the Aho and Ullman memory organization. There are $N = r^m - 1$ cells in the memory numbered 0 through $N-1$. Location 0 is the i/o port. Assume that m can be written as the product of two integers c and w ; i.e., $m = cw$. The memory cells are considered to constitute one set. Two memory permutations are defined as follows.

$$d_{s,w}^r(j) = (r^{(c-1)w}j) \bmod N$$

$$d_c(j) = j-1 \bmod N, \quad 0 \leq j < N.$$

Note that the base r representation of $d_{s,w}^r(j)$ is the left rotation by w r -ary digits of the base r representation of j . The permutation $d_{s,w}^r$ can be referred to as the generalized multiple shuffle.

The permutations generated for the random access of datum j are defined by the access algorithm in Figure C.1. The algorithm is similar to the Aho and Ullman access algorithm. One change is that the permutation d_c is used to zero a block of w positions. The permutation $d_{s,w}^r$ rotates the base r representation of an address by w positions. Note that Aho and Ullman use left rotations while Wong and Tang use right rotations.

Input: j (datum to be accessed) and $s(\cdot)$ (the memory map);

Output: Access Sequence

- (1) Let a be the m digit base r representation of $s(j)$. Divide a into c blocks of w digits each. Number the blocks $1, 2, \dots, c$ from right to left. Let the numbers represented by the blocks be $a_1, a_2, \dots, a_c$ respectively.
- (2) $k \leftarrow 1$.
- (3) Apply $a_k d_c$'s to the memory. $a_k \leftarrow 0$.
- (4) If $a_i = 0$ for all $i > 1$, stop.
- (5) Apply $d_{s,w}^r$ to the memory.
- (6) $k \leftarrow k+1$. Go to (3).

Access Algorithm for the Wong-Tang Two Permutation Memory

Figure C.1

Wong and Tang have calculated the worst case access time to be $\frac{r^w}{w} m - 2$, i.e., $\frac{r^w}{w} \log_r(N+1) - 2$ or $\frac{r^w}{w \log r} \log(N+1) - 2$. This quantity can be minimized by the choice of r and w . Such an optimization has been carried out by Wong and Tang and the results are given in Table C.1. $r=3$ and $w=1$ minimizes the worst case access time to $1.893 \log(N+1) - 2$. Note that the results for the Aho and Ullman memory can be obtained from Table C.1 by setting $w = 1$.

Assume that the probability of datum j being accessed is uniform for j , over the address range $(0, N-1)$. Wong and Tang have evaluated the average random access time to be approximately $\frac{r^{w+1}}{2w} m - \frac{r^w}{r^w - 1}$;

i.e., $\frac{r^{w+1}}{2w \log r} \log(N+1) - \frac{r^w}{r^w - 1}$. From the results given in Table C.1

note that $r=2, w=2$ or $r=4, w=1$ gives the minimum average random access time of $1.25 \log(N+1) - 1.333$.

r	2	3	4	5
w				
1	$2\hat{m}-2$	$1.893\hat{m}-2$	$2\hat{m}-2$	$2.153\hat{m}-2$
2	$2\hat{m}-2$	$2.839\hat{m}-2$	$4\hat{m}-2$	$5.383\hat{m}-2$
3	$2.667\hat{m}-2$	$5.678\hat{m}-2$	$10.667\hat{m}-2$	$17.945\hat{m}-2$
4	$4\hat{m}-2$	$12.776\hat{m}-2$	$32\hat{m}-2$	$67.293\hat{m}-2$

Worst Case Random Access Time

$$\frac{r^w}{w \log r} \log(N+1) - 2$$

$$\hat{m} = \log(N+1)$$

Table C.1

The following discussion pertains to the worst and average case block access time for the Wong and Tang memory. Wong and Tang show that $s(j)$ always has the form $s(j) = (r^{wp}j + q) \bmod N$ for integers p and q where $0 \leq p < c$ and $0 \leq q < m$. Note that

$$d_c \text{ maps } \begin{cases} p \text{ into } p \\ q \text{ into } (q-1) \bmod N \end{cases}$$

and

$$d_{s,w}^r \text{ maps } \begin{cases} p \text{ into } p-1 \\ q \text{ into } (r^{(c-1)w}q) \bmod N \end{cases}.$$

Let datum k be at the head of the block. Immediately after bringing datum k to the i/o port, location 0, $s(k) = r^{wp}k + q \bmod N = 0$.

$s(k+1) = r^{wp}(k+1) + q \bmod N = r^{wp} \bmod N$. Application of $p d_{s,w}^r$

permutations on the memory followed by a d_c brings datum $k+1$ to the

i/o port. At the termination of the access to datum $k+1$, in this fashion, the memory map is given by $s(j) = r^0j + q'$. Since $s(k+1)$ is now 0, $k+1 + q' = 0 \bmod N$ and $q = -(k+1) \bmod N$. Subsequent accesses for the block take unit time per access through the application of one d_c for every access.

Assume that block accesses start at logical addresses that are uniformly and independently distributed over the address range $(0, N-1)$. The worst case access time for datum $k+1$ is $\frac{m}{w} = \frac{\log_r(N+1)}{w}$. The worst case block access time for a block of b items is

$\frac{r^{w+1}}{w \log r} \log(N+1) - b - 4$. This quantity has been evaluated for

several values of r and w . The results are presented in Table C.2. The minimum worst case block access time of $2.500 \log (N+1) + b - 4$ is obtained for $r=4, w=1$ or $r=2, w=2$.

To evaluate the average block access time, additional assumptions must be made, as in the case of the Aho and Ullman memory. Assume that all memory requests are block access requests for blocks of at least size 2. Assume that at the beginning of every block access, data items are sequentially arranged in the memory according to their logical addresses. This assumption is met if either the memory is heavily loaded or if whenever the memory is idle d_c permutations are applied to its contents. As demonstrated for the Aho and Ullman memory, starting addresses may be divided into classes. Let k denote the starting address of the block. For all addresses in a class the access time for datum $k+1$ is the same. Table C.3 describes the general form of the addresses within a class, the number of $d^{r,w}$ permutations performed to access datum k for all

r	2	3	4	5
w				
1	$3\hat{m}-2+b$	$2.524\hat{m}-2+b$	$2.500\hat{m}-2+b$	$2.584\hat{m}-2+b$
2	$2.500\hat{m}-2+b$	$3.154\hat{m}-2+b$	$4.250\hat{m}-2+b$	$5.598\hat{m}-2+b$
3	$3\hat{m}-2+b$	$5.888\hat{m}-2+b$	$10.834\hat{m}-2+b$	$18.008\hat{m}-2+b$
4	$4.250\hat{m}-2+b$	$12.934\hat{m}-2+b$	$32.126\hat{m}-2+b$	$67.4\hat{m}-2+b$

Worst Case Access Time for a Block of b Items

$$\frac{r^w + 1}{w \log r} \log (N+1) - 2 + b$$

$$\hat{m} = \log (N+1)$$

Table C.2

members in the class, the time taken to access datum $k+1$ for each class, and the number of members in each class. In the first column of Table C.3, each position in the address represents a base r number formed of w base r digits. Here 'n' represents a non-zero number and '*' represents any number in the range $(0, r^w-1)$.

Proceeding as in the case of the Aho and Ullman memory the average time for the access of datum $k+1$ is evaluated to be

$$\frac{r^w - 2 + \frac{m}{w}(r^m - r^w) - \frac{r^{2w}}{r^w - 1} \left[\left(\frac{m}{w} - 2 \right) r^{m-w} - \left(\frac{m}{w} - 1 \right) r^{m-2w} + 1 \right]}{r^m - 1},$$

which for $r^m \gg r^w$, is approximately $\frac{2r^w - 1}{r^w - 1}$. The average access time

for a block of b items is given by $\frac{r^{w+1}}{2^w} - \frac{r^w}{r^w - 1} + \frac{2r^w - 1}{r^w - 1} + b - 2$,

assuming $r^m \gg r^w$. This simplifies to $\frac{r^{w+1}}{2^w} \log_r(N+1) + b - 1$. From

<u>Address Form</u> <u>of class</u>	<u>Num. of right</u> <u>rotations for</u> <u>access of k</u>	<u>Time for</u> <u>access of</u> <u>k+1</u>	<u>Num. of</u> <u>members in</u> <u>the class</u>
00 .. 000	0	1	1
00 .. 00n	0	1	$r^w - 1$
00 .. 0n*	1	$\frac{m}{w}$	$(r^w - 1)r^w$
00 .. n**	2	$\frac{m}{w} - 1$	$(r^w - 1)r^{2w}$
..	.	.	.
..	.	.	.
0n .. ***	$\frac{m}{w} - 2$	3	$(r^w - 1)r^{(\frac{m}{w} - 2)w}$
n* .. ***	$\frac{m}{w} - 1$	2	$(r^w - 1)r^{(\frac{m}{w} - 1)w - 1}$

Table C.3

Table C.2 it is determined that $r=2$, $w=2$ or $r=4$, $w=1$ minimize the average block access time to $1.25 \log (N+1) + b - 1$.

Modifications to the Wong and Tang memory analogous to those proposed by Stone for the Aho and Ullman memory may be made. This involves renumbering the memory locations from 1 to $r^m - 1$. This modification can be carried out because the $d^{r, w}$ permutation can be shown to be an automorphism for r 's complement arithmetic. The modification results in simplification of the access algorithm but no significant change in average access times. A reduction in worst case block access time is expected.

APPENDIX D

The Three Permutation Wong-Tang Memory

Wong and Tang have proposed a memory organization [10] with three permutations. The first two are the same as defined in their earlier organization. The additional permutation is

$$d_a(i) = (i+1) \bmod N.$$

Wong and Tang also give the access algorithm for this organization. The algorithm is similar to the access algorithm for the two permutation memory. However, in this algorithm either d_c or d_a can be used to zero the set of w low order digits of the current address of any datum. For each set that permutation is chosen that requires fewer applications. The access algorithm takes into account the problem of carry propagation when using the d_a permutation.

For purposes of analysis it is necessary to consider memories for which r is even separately from memories for which r is odd. When r is even, d_c and d_a are used with equal frequency. This makes average and worst case random access time analysis tractable, under the usual assumptions that the probability of random access is uniform over the address range. For even values of r the worst case random access time is $\frac{r^{w+2}}{2w \log r} \log (N+1) - 2$, for $c = \frac{m}{w}$ odd, and is $\frac{r^{w+2}}{2w \log r} \log (N+1) - 1$, for c even.

The worst case random access time for odd r is found to be

$\frac{r^{w+1}}{2w \log r} \log (N+1) - 1$. The value of the coefficient of the logarithmic term $\log (N+1)$ is shown in Table D.1 for different values of w and r . $r=3$ and $w=1$ minimizes the worst case random access time to $1.262 \log (N+1) - 1$.

In evaluating the worst case block access time, recall Wong and Tang's observation that the memory map can be represented by $s(j) = r^{wp}j + q \pmod N$, with $0 \leq p < \frac{m}{w}$ and $0 \leq q < N$. Arguments similar to those advanced in Appendix C can be used to show that if at the end of the access to datum k the memory map is of the form $r^{wp}k + q$, then application of $p d_{s,w}^r$ and one d_c permutation brings datum $k+1$ into the i/o port (location 0). Subsequent accesses takes unit time per access. The worst case block access time is the worst case random access time + $\frac{\log (N+1)}{w \log r} + b - 2$. The coefficient of the term $\log (N+1)$ is derived from Table D.1 and is evaluated in Table D.2, for different r and w . The minimum worst case block access time is

r	2	3	4	5	6	7	8
w							
1	2	1.262	1.5	1.292	1.547	1.425	1.667
2	1.5	1.577	2.25	2.799	3.675	4.453	5.5
3	1.667	2.944	5.5	9.044	14.055	20.425	28.556
4	2.25	6.467	16.125	33.7	62.766	106.965	170.75

Coefficient of $\log (N+1)$ in worst case random access time

$$r \text{ even} : \frac{r^{w+2}}{2w \log r} \quad r \text{ odd} : \frac{r^{w+1}}{2w \log r}$$

Table D.1

r	2	3	4	5	6	7	8
w							
1	3	1.893	2	1.723	1.934	1.781	2
2	2	1.892	2.5	3.014	2.868	4.631	5.667
3	2	3.145	5.667	9.188	14.184	20.544	28.667
4	2.5	6.625	16.25	33.808	62.863	107.054	170.833

Coefficient of $\log (N+1)$ for the worst case
block access time for a block of b items.

$$r \text{ even} : \frac{r^w+4}{2w \log r} \quad r \text{ odd} : \frac{r^w+3}{2w \log r}$$

Table D.2

$1.723 \log (N+1) + b - 3$ for $r=5, w=1$.

Wong and Tang evaluate the average random access time for even values of r to be $\frac{r^m}{r^m-1} \left[\frac{r^w+4}{4w} - \frac{r^w-r^{-m+w}}{r^w-1} \right]$. When $r^m \gg r^k$, this can be approximated by $\frac{r^w+4}{4w \log r} \log (N+1) - \frac{r^w}{r^w-1}$. When r is odd the

permutations d_c and d_a are not used with equal frequency. However,

Wong and Tang have analyzed the approximate average random access time assuming that these probabilities are indeed equal. The average random access time for large values of m is calculated to be

$$\frac{r^w+4}{4w \log r} \log (N+1) - \frac{1}{4r^w(r^w-1)} - \frac{r^w}{r^w-1}. \quad \text{The minimum value for}$$

$\frac{r^w+4}{4w \log r}$ can be found in Table D.3 to be $0.967 \log (N+1) - 1.2$ for $r=6$ and $w=1$.

To evaluate the average block access time, assume that data items in memory are laid out sequentially at the beginning of each block access. As seen in Appendix C this can be ensured by the

r	2	3	4	5	6	7	8
d							
1	1.5	1.104	1	0.969	0.967	0.980	1
2	1	1.025	1.25	1.561	1.934	2.360	2.833
3	1	1.630	2.833	4.630	7.092	10.302	14.333
4	1.25	3.352	8.125	16.931	31.431	53.549	85.417

Coefficient of $\log (N+1)$ in the average random access time

$$\frac{r^w + 4}{4w \log r}$$

Table D.3

heavily loaded hypothesis or by the specification of memory behavior for idle periods. Again the address space is divided into several classes. Let the block access begin with access to datum k . For all k belonging to one of these classes, access to datum k and access to datum $k+1$ can take one of two times each. This is due to the possibility of a carry being generated from the high order non-zero w digits into the next set. This results from the application of the d_a permutation to clear the high order non-zero w positions. Wong and Tang show that the probability of the generation of a carry from one set of w positions into another is $1/2$ when r is even. When r is odd, this is not true, but for the purposes of approximate analysis, this probability is assumed to be $1/2$ for odd values of r as well.

The access algorithm partitions the set of all addresses into two disjoint sets. Access strategies are slightly different for these two sets. However, each address in one set has a corresponding address, called its analog, in the other that has the same access sequence expect that the roles of d_a and d_c are interchanged. The

two addresses are diminished radix complements of each other.

Table D.4 illustrates the address classes. The number of $d_{s,w}^r$'s used for the access of any datum k in the class is tabulated along with the access time for datum $k+1$. Each class consists of addresses and their analogs. A random address belonging to any but the first or last class has two equiprobable access times. These are indicated in the table. The average access time for a class is the mean of these two access times. The sum of these class access times weighted by the frequency of the class and normalized by the total number of addresses in the address range is the average access time for datum $k+1$ during block access. This is given by $\frac{u}{r^m-1}$, where

$$u = \frac{1}{r^w-1} [2r^{m+w} - 2r^w] - \frac{m}{w} [r^w - 1] - r^{m-w}.$$

When $r^m \gg r^w$ $\frac{u}{r^m-1}$ can be approximated by $\frac{2r^w}{r^w-1} - \frac{1}{r^w}$.

The average block access time when $r^m \gg r^w$ is approximated by

$$\frac{r^{w+4}}{4w \log r} \log (N+1) - \frac{r^w}{r^w-1} \frac{2r^w}{r^w-1} - \frac{1}{r^w} + b - 2.$$

for even r , and

$$\frac{r^{4+4}}{4w \log r} \log (N+1) - \frac{1}{4r^w(r^w-1)} + \frac{r^w}{r^w-1} - \frac{1}{r^w} + b - 2.$$

when r is odd. From Table D.3 the minimum average block access time is $0.967 \log (N+1) - 0.97 + b$ for $w=1$, $r=6$.

<u>Address</u> <u>Form</u>	<u>Number Of Multiple</u> <u>Shuffles To Access</u> <u>Datum k</u>	<u>Time To Access</u> <u>$k+1$</u>	<u>Frequency Of</u> <u>Class</u>
00 .. 00	0	1	1
00 .. 0n	0	1	$2(r^w-1)$
	1	$\frac{m}{w}$	
	1	$\frac{m}{w}$	
00 .. n*	2	$\frac{m}{w} - 1$	$2(r^w-1)r^w$
..	..	..	..
..	..	..	..
	$\frac{m}{w} - 2$	3	
0n .. **	$\frac{m}{w} - 1$	2	$2(r^w-1)r^{w(\frac{m}{w}-2)}$
v* .. **	$\frac{m}{w} - 1$	2	$2(\frac{r^w}{2}-1)r^{w(\frac{m}{w}-1)}$
	$0 \leq * < r^w$	$0 < n < r^w$	$0 < v < \frac{r^w}{2}$

Table D.4

REFERENCES

- [1] W. Kluge, "A Dynamic Memory with Fast Random Access and Page Transfer Properties," Digit Process Vol. 1(4) (1975).
- [2] D. P. Bhandarkar and J. E. Juliussen, "Tutorial: Computer System Advantages of Magnetic Bubble Memories," COMPUTER, pp.35-40 (November 1975).
- [3] H. S. Stone, "Dynamic Memories with Enhanced Data Access," IEEE Trans. Computers Vol. C-21(4) (April 1972).
- [4] K. E. Batchter, "Apparatus for Shifting Data in a Long Register," United States Patent 3,605,024 (June 1, 1970).
- [5] C. K. Wong and D. Coppersmith, "The Generation of Permutations in Magnetic Bubble Memories," IEEE Trans. Computers Vol. C-25(3), pp.254-262 (March 1976).
- [6] H. S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Trans. Computers Vol. C-20(2) (February 1971).
- [7] A. V. Aho and J. D. Ullman, "Dynamic Memories with Rapid Random and Sequential Access," IEEE Trans. Computers Vol. C-23(3), pp.272-276 (March 1974).
- [8] H. S. Stone, "Dynamic Memories with fast Random and Sequential Access," IEEE Trans. Computers Vol. C-24(12), pp.1167-1174 (Dec 1975).

- [9] C. K. Wong and D. T. Tang, "Dynamic Memories with faster Random and Sequential Access," IBM Journal of Research and Development Vol. 21(3), pp.281-288 (May 1977).
- [10] C. K. Wong and D. T. Tang, "Dynamic Memories with Faster Random and Sequential Access," IBM Research Report (October 1975).
- [11] J. Lenfant, "Fast Random and Sequential Access to Dynamic Memories of any size," IEEE Trans. Computers Vol. C-26(9), pp.847-855 (Sept. 1977).
- [12] H. Fredrickson, A Note on Stone's Dynamic Memory (Unpublished).
- [13] S. B. Morris, A. Valiere III, and R. A. Wisniewski, "Processes for Random and Sequential Accessing in Dynamic Memories," IEEE Trans. Computers Vol. C-28(3), pp.225-237 (March 1979).
- [14] D. Hoey and C. E. Leiserson, "A Layout for the Shuffle-Exchange Network," CMU-CS-80-139, Dept. of Computer Sc., Canegie-Mellon University (18 August 1980).
- [15] H. H. Loomis Jr., "Data Representation," pp. 23-75 in Introduction to Computer Architecture, ed. H. S. Stone, SRA Inc., Chicago (1975).