

CONCEPTS AND CAPABILITIES OF DATABASE MACHINES

by

NASSRIN TAVAKOLI

B.S., Southeast Missouri State University, 1978

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements of the degree

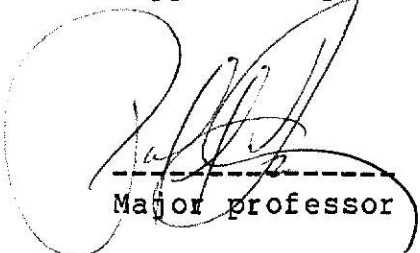
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1981

Approved by:



Major professor

SPEC
COLL
LD
2668
R4
1981
T38
C.2

A11200 395480

i

ACKNOWLEDGEMENTS

I wish to express my sincere thanks to my advisor, Dr. Paul S. Fisher, for his valuable help, guidance, and suggestions. Thanks are also given to the other members of my committee, Dr. Elizabeth A. Unger and Dr. Rodney M. Bates.

TABLE OF CONTENTS

1.0	Introduction	1
2.0	Limitations of Conventional Computers for database applications	4
2.1	Mismatch of Conventional Computers for Database Applications	4
2.2	Use of Physical Addressing	4
2.3	Many Levels of Mapping	5
2.4	Performance Bottlenecks	5
2.5	User's Increasing Demand for DBMS Capabilities	6
3.0	Database Machines (Architecture, Objectives, and Characteristics	8
3.1	Cellular_Logic Systems	8
3.1.1	Some Capabilities of Cellular Logic Systems	12
3.1.2	Some Limitations of Cellular Logic Systems	14
3.1.3	CASSM _ Context Addressed Segment Sequential Memory	16
3.1.4	RAP _ Relational Associative Processor	27
3.1.5	RARES _ Rotating Associative Relational Store	34
3.2	High Speed Associative Memory Systems	37
3.2.1	Some Capabilities of	

Associative Devices	41
3.2.2 Some Advantages of Associative Devices	41
3.2.3 Some Disadvantages of Associative Devices	42
3.2.4 STARAN	43
3.3 Backend Computers	52
3.3.1 Advantages	54
3.3.2 Disadvantages	57
3.3.3 XDMS _ Experimental Data Management System	58
3.3.4 DC _ Data Computer	63
3.3.5 IDM _ Intelligent Database Machine	68
3.4 Integrated Database Machines	71
3.4.1 Advantages	72
3.4.2 Disadvantages	72
3.4.3 DBC _ Data Base Computer	73
3.4.4 DIRECT	78
4.0 The Future of Database Machines	84
List of References	87
Additional readings	90

LIST OF FIGURES

1	The Architecture of a Cellular_Logic device	9
2	Overall Architecture of CASSM	18
3	General view of a CASSM cell	21
4	CASSM word types	24
5	Overview of RAP Architecture	28
6	Overview of RAP Cell Architecture	30
7	Track Format	32
8	RARES data organization	36
9	A typical Associative Memory System Configuration . .	37
10	The Architecture of an Associative Memory	39
11	STARAN System Configuration	45
12	Associative Module	47
13	Control System	49
14	A Configuration of a Backend Computer System . . .	53
15	Mudole Host Configuration	55
16	Multiple Backend Configuration	56
17	XDMS Hardware Configuration	60
18	Logical View of Data Computer	64
19	Hardware Overview of system	66
20	IDM Configuration	69
21	The Architecture of DBC	74
22	DIRECT System Architecture	80

CHAPTER 1

INTRODUCTION

The majority of DBMS's in use today are implemented on conventional von Neumann computers. However, conventional computers are not well-suited and efficient for performing DBMS functions, mainly because they are not highly capable of storing and retrieving information. In addition the processing and storage areas of these systems are inefficient. This raises the need for a specialized hardware device capable of handling DBMS primitives (e.g. storage, retrieval, update, etc.) more efficiently and cost-effectively; namely, a data base machine.

In the past few years there has been increased emphasis and research in the area of data base machines, and there are several reasons behind these increased research activities:

1. There is an increase in non-numeric applications such as database management.
2. The change of data-processing-oriented information management to database-management-oriented information management, and the multiple users accessing the shared database. This requires considerable software development and hardware

support.

3. The increasing need for larger databases. Very large databases complicate the problem of retrieval, update, data recovery, transaction processing, integrity and security. The software solutions to these problems become very complex and costly. Therefore, an alternative hardware solution to these problems becomes essential.
4. The improvement of user/programmer productivity, and the need for protection of applications from changes in the user environment suggests more powerful database management systems which support the high-level data model and language. However, supporting these interfaces by means of software becomes inefficient and complex because of mapping the high-level data representation and languages to low-level storage representation and machine codes.
5. The availability and variety of memory and processor technology such as charge-coupled devices, electron-beam laser memories, magnetic bubble memories, modifiable moving head disks, associative devices, etc., coupled with the fact that the cost of electronic hardware devices are drastically dropping.

6. Increases in cost of software and personnel.
7. Vigorous drive toward DBMS Standards led by National Bureau of Standards (NBS) aiming to [25] "1) protect the federal investment in existing data, programs, and personnel skills, 2) improve the productivity and effectiveness of database systems available to federal agencies, 3) assist federal agencies with selection, procurement, use, and availability of database systems, 4) perform the research necessary to identify future federal needs and to foster the development of necessary database tools." 1

The impact of standards must be considered when contemplating development of a data base machine.

This paper begins by characterizing the major problems and limitations of conventional systems. This is followed by a review of the existing database machines, their objectives, and their characteristics. In the last section a few thoughts are presented on future trends.

CHAPTER 2

LIMITATIONS OF CONVENTIONAL COMPUTERS FOR DATABASE APPLICATIONS

2.1 Mismatch of Conventional Computers for Database Applications

Conventional computers were originally designed for numeric applications such as add, shift, etc.; whereas most applications today are non-numeric. This mismatch of the conventional computer design to non-numeric applications is the main cause of the complexity and inefficiencies of these systems. Most primitive operations required by database systems must be performed by software routines, which not only execute slower than hardware, but are less reliable, more expensive to maintain, and harder to test (or prove).

2.2 Use of Physical Addressing

The use of physical addressing by conventional systems generate significant overhead in a DBMS, where the retrieval of information is the primary function. As a result, searching and manipulating data in large databases

is too slow to meet the response time requirement of many applications. Although software techniques such as hashing functions, directories, pointers, inverted files, and indexes alleviate the speed problem to a certain extent,

they introduce understandable side effects such as excess storage requirements and overloading of the CPU -- due to the many software routines needed to support these techniques.

2.3 Many Levels of Mapping

The recent effort toward more powerful database management systems requires the use of high-level data models and high-level languages, which increase data independence and improve human productivity. However, the implementation of these high-level data models and data languages require many levels of complex software. These levels results in inefficiencies in system utilization and response. The software complexity and system inefficiencies are due to the mapping of high-level commands and data views into low-level machine codes and structures.

2.4 Performance Bottlenecks

One of the problems with conventional systems is the "staging" problem. A conventional computer can only manipulate data which are stored in main memory. Since the amount of main memory is typically smaller than the size of the database , large amounts of data must be stored on secondary storage and then moved into main memory for processing. This transformation of data between different

storage hierarchies is called "staging". The process of staging, however, is time consuming and degrades the performance of the entire system; for example, using the IBM 3300 disc pack, which has a transmission rate of 806 character/msec [24], would take several seconds just to transfer a large file into main memory.

Staging also causes data communications over long distances to be expensive and limited in speed. As a result, database systems have to physically distribute data where usage is highest. A great amount of redundancy, therefore, is often purposefully introduced to avoid excess amounts of data transfer and to improve performance and reliability. This distribution of data also causes problems in updating the data, in recovering from system failures, and in the integrity and security of the data. To "stage" the data in the main memory frequently ties up the communication lines and channels and the database as well. This is the primary reason why the conventional DBMS is often CPU-bound and short of main memory cycles.

Ideally, data should be processed at the place where it is stored to avoid spending time in moving excess data between the various levels of memory.

2.5 User's Increasing Demands for DBMS Capabilities

The user's demand for more sophisticated DBMS capabilities are continuously increasing. Capabilities such

as automatic database restructuring and system tuning, automatic data distribution and redistribution, integrity and security controls, back up and recovery, etc., are handled by software in conventional systems. The result is a tremendous overhead and complexity in implementing these capabilities.

CHAPTER 3

DATABASE MACHINES

(Architecture, Objectives, & Characteristics)

The existing database machines fall into different categories based on their architectural taxonomy and their differences in objectives and characteristics. In this section each category and the relevant database machines are introduced.

3.1 Cellular-logic Systems

The basic idea of cellular-logic systems is to move some of the frequently used database management functions to "intelligent" secondary storage devices so that these functions can be carried out by the storage devices without the attention of the main processor [24,25].

This is done by building more processing capabilities into secondary storage devices (disks, drums, CCD's or magnetic bubble) so that data can be searched and partly processed at the secondary storage site. Irrelevant data can, therefore, be filtered out before moving to main storage. Processing efficiency is also increased by parallel processing of the data in the memory elements and by doing content and context addressing of the data. The general architecture of a cellular-logic system is illustrated in

Figure 1.

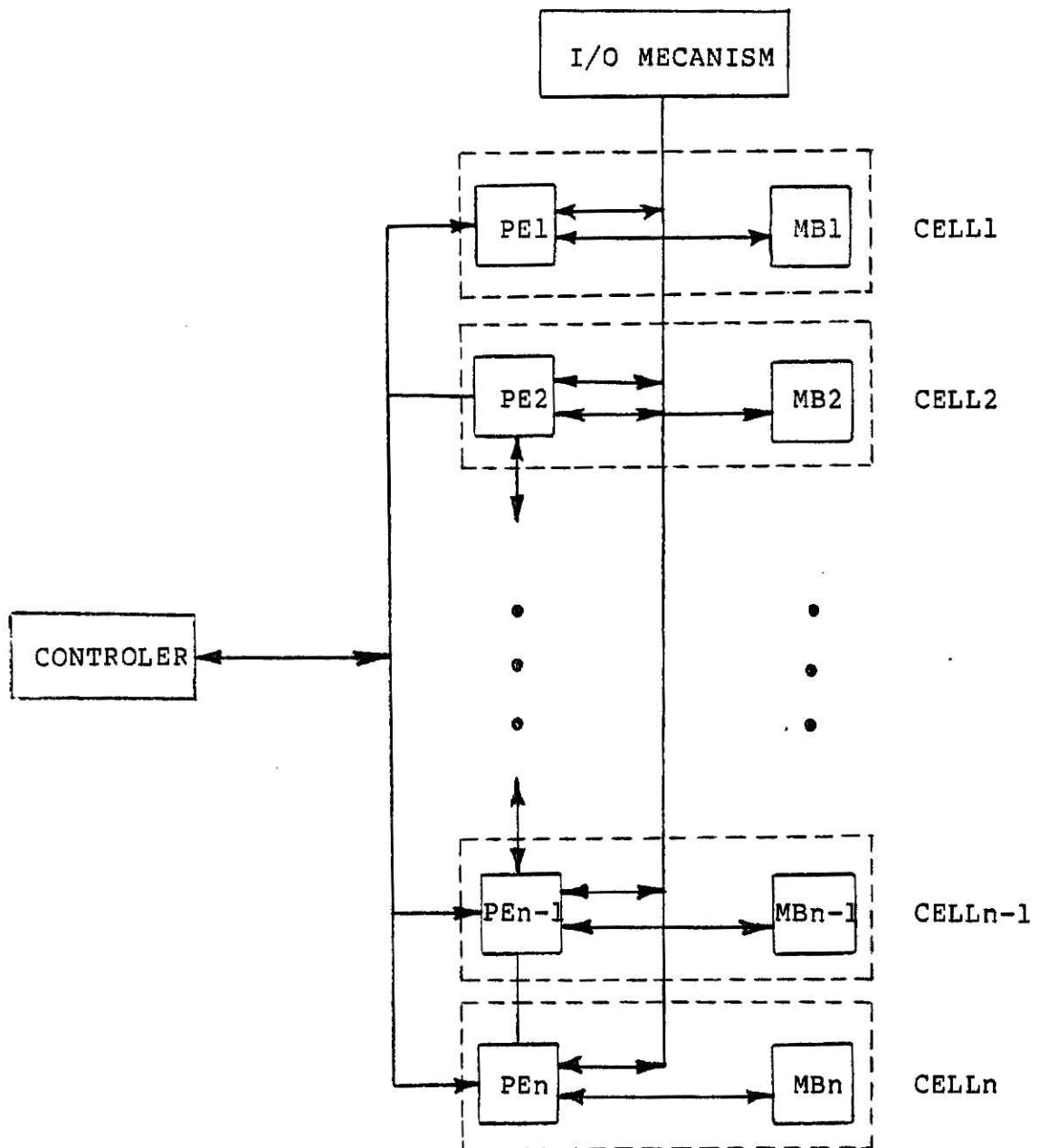


Figure 1. The Architecture of a Cellular-logic Device

A cellular-logic system consists of four major components: memory blocks, processing elements (these two are included in one cell), a controller, and an I/O mechanism.

Memory Blocks

Memory blocks (MB) are storage facilities such as disk tracks, magnetic bubble memories, or charge-coupled devices. They are used to store three types of information: data, control information (e.g. tag fields, status fields, etc.), and executable instructions. This information is serially accessible within each block.

Memory blocks are generally cheaper per bit than memories that allow random addressing at the character level. Memory blocks will generally be classified as slow access, however, they are slow only when used to emulate a random access memory. When used for the exhaustive associative search, they are as efficient as a random access memory. In addition to searching efficiency, these devices offer efficient storage management for update. Also, supportive data structures such as indexes, pointers, hash tables, etc., are eliminated and the effective storage cost per bit is further reduced. In summary, the block serial nature of these devices can be fully exploited to improve simplicity, efficiency, and data independence [24,25].

Processing Elements

Processing elements (PE's) are small processors which

are capable of performing primitive non-numeric functions. there is a one-to-one relationship between processors and memories which allows an efficient utilization of both.

Each of the processors contains a small buffer which is used to hold search arguments and data from the memory blocks. They are capable of performing most or all of the following types of functions [24]:

1. Comparisons
2. Logical operations (AND, OR, NOT)
3. Arithmetic operations ($=$, $>$, \geq , $<$, etc.)
4. Primitive statistical operations (AVERAGE, SUM, MAX, MIN, COUNT, etc.)
5. Primitive database operations (search, retrieve, insert, delete, modify, sort)

The Controller

The major function of the controller is to broadcast instructions to the processing elements. Instructions may either be stored within memory blocks, or they may be fed to the controller by the host computer (or front end). The instruction is broadcast simultaneously to all the PE's for execution. It operates on different data streams stored in the MB's. Thus, cellular-logic systems can be classified as having the SIMD (Single Instruction stream, Multiple Data stream) architecture.

The I/O Mechanism

The I/O mechanism is used to transfer information in and out of the memory blocks. The transfer of information from multiple MBs can be done serially, or in parallel, depending upon the capability of the I/O channel.

3.1.1 Some Capabilities of Cellular-logic Systems

Parallel Processing

The multiple processing elements of a cellular-logic device (CLD) operate concurrently, thereby providing a parallel processing capability. A database operation such as search, retrieve, update, delete, or insert is broadcast simultaneously to all processors which carry out the operation against the data residing in their associated memory elements. Thus, in one rotation of the memory, the entire database is reached in $1/n$ (th) of the time needed for a sequential search over n segments of data [5,16].

Content and Context Addressing

Since the entire database is searched in each circulation of the memory, data can either be searched associatively by content or context addressing, rather than the physical addressing used by conventional systems.

The content addressing scheme addresses memory by the value of the stored data items. In order to access a particular record, the value of one of the fields is

specified (part of the field can also be use as a search argument). Then, by searching through each of the records in the file, the record(s) which contain the specified information are located.

The context addressing scheme also addresses memory by content, but in addition, the search is controlled by the results of one or more previous searches. As an example, consider the retrieval of a set of employee records which satisfy the following two conditions: Name = "JONES" and Salary < 25,000. A content search can be used to locate all of the records which satisfy the first condition. Then, with the context of this result, the set of records which satisfy both conditions is obtained by checking these records against the second condition. In order to evaluate more complex expressions, it may be necessary to store the results of several previous content or context searches. Therefore, several bits of additional storage are required for each record in the database. But this is offset by the elimination of the special supportive structures such as indexes, hash tables, pointers, etc. used in the conventional systems. The content and context search techniques in the CLD's, therefore, offer uniformity and fast response time for search and update operations.

Statistical and Database Primitives

In addition to the content and context searching capability, CLDs provide hardware support for a variety of

other primitive statistical and DBMS functions such as update, parity checking, garbage collection, evaluation of boolean expressions, and arithmetic functions such as: MIN, MAX, SUM, COUNT, etc. [7].

3.1.2 Some Limitations of Cellular-logic Devices

Capacity

CLDs are currently too expensive to support large databases.

Data Types

CLD hardware will only recognize two types of data: character strings and integers [24].

Numerical Processing Capabilities

CLDs have very limited numerical processing capabilities. CASSM, for example, does not support multiplication or division [7], and CLDs in general are limited to fixed point arithmetic [24].

Other functions

CLD's do not have facilities that will support other functions, such as system recovery, integrity, security control, etc.

Processor Utilization

Most of the proposed CLD designs (including CASSM and RAP) [19] are single instruction multiple data stream (SIMD) architectures. That is, a single instruction is simultaneously executed by all of the processing elements. This approach is simple to implement, but it does not allow an efficient utilization of the processing elements, since it is likely that only a few of the memory blocks will contain information that is relevant to any given query. An alternative to the SIMD architecture, called multiple instruction multiple data stream (MIMD), is designed to improve the utilization of the processing elements by allowing the simultaneous execution of multiple instructions by disjoint sets of processing elements. An example of MIMD architecture is DIRECT, which is currently being implemented at the University of Wisconsin [12].

Intercell Communication

Communication between the cells of a CLD is required in order to keep a file in sequential order, and to allow a single record to be split over two or more cells [12,24]. However, an intercell communications network is difficult and expensive to implement, and should, therefore, be kept simple [1].

In summary, the distinguishing features of the cellular-logic approach are: 1) increased processing capabilities in secondary storage devices, 2) search time is independent of the database size, 3) elimination of the need

for building, updating, and protecting auxiliary structures, 4) the use of identical cells to increase reliability, flexibility in adding or reducing the number of cells and to reduce the cost of production, 5) the potential for extremely high speeds as cell sizes decrease and memory density and speed increase (i.e. increase in the ratio of processing power to memory).

Several systems have been designed based on the cellular-logic approach and some have gone through prototype implementation. Some of these systems are discussed here.

3.1.3 CASSM - Context Addressed Segment Sequential Memory

The CASSM [23,26] project began at the University of Florida in 1972. The hardware primitives were designed to carry out the basic operations required for supporting the retrieval and manipulation of hierarchically structured data files. However, since relations in the relational model can be considered as two-level trees with the relation name as the root and the tuples as leaves, and the network represented by a family of trees, the CASSM hardware can also be used to support relational and network database management. Due to its head-per-track architecture, CASSM is practical only for databases smaller than 10^8 bytes in size.

Overall Architecture

The overall architecture of CASSM system is illustrated in Figure 2.

**THIS BOOK
CONTAINS
NUMEROUS PAGES
WITH DIAGRAMS
THAT ARE CROOKED
COMPARED TO THE
REST OF THE
INFORMATION ON
THE PAGE.**

**THIS IS AS
RECEIVED FROM
CUSTOMER.**

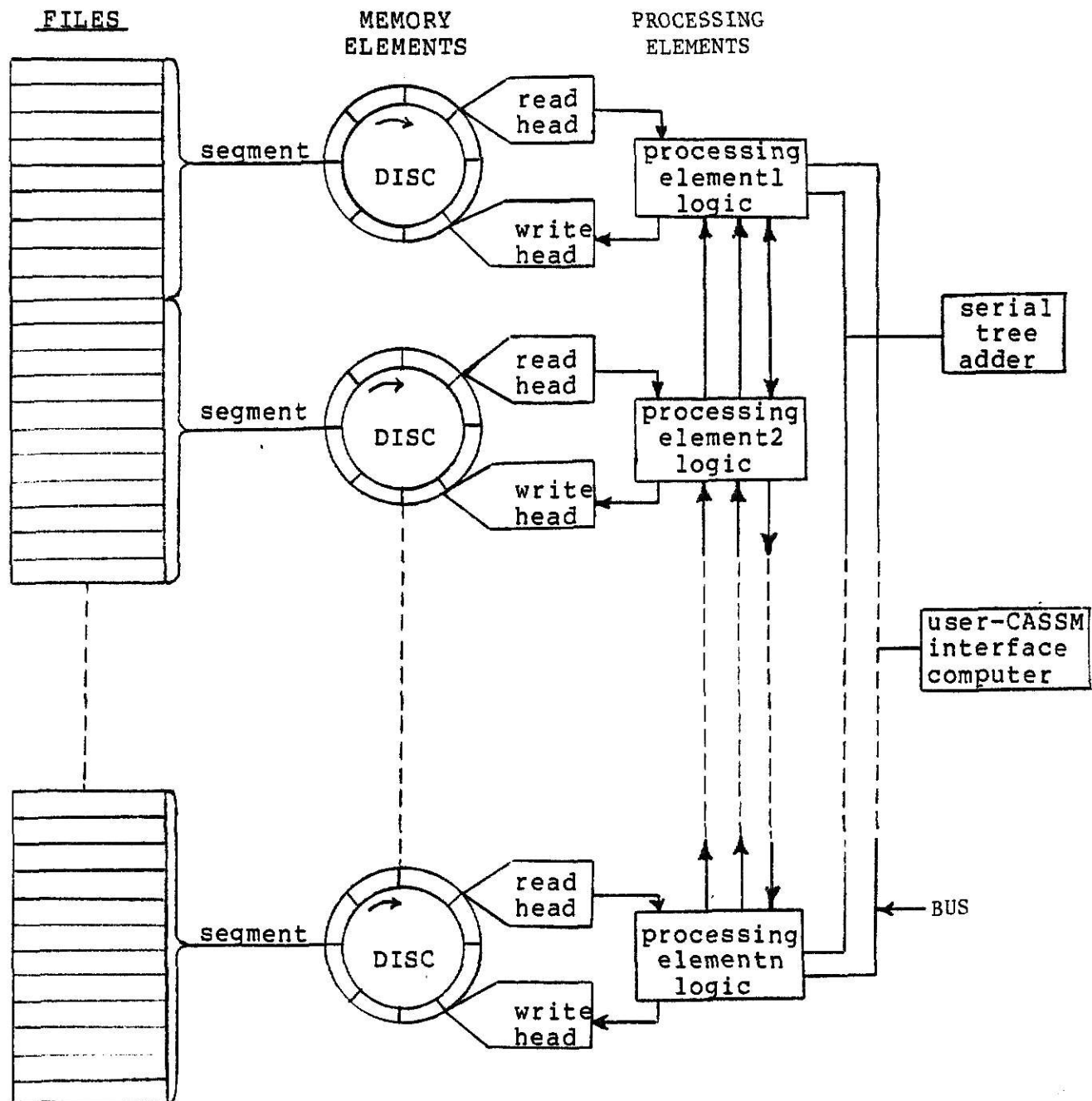
Data and program

Figure 2. Overall architecture of CASSM

Data as well as the compiled programs are stored in the memory elements and are manipulated by the associated processing elements. One particular feature of CASSM hardware is that the processing elements can directly communicate with their adjacent neighbors, this intercell communication provides the necessary support to allow sequential files to be segmented and stored sequentially in the memory elements starting from the first one to the nth. Logically, a file is a collection of records which contain hierarchically structured data and/or instructions of CASSM's machine language program. Physically, files are partitioned into segments each of which resides in a memory element. The segments are simultaneously processed by the processing elements. Data or instructions are read by a read head from the circular memory element, processed by the processing element (a special purpose microprocessor), and written back to the memory element by a separate write head. Data is passed among the processing elements and the interface computer through a data bus. A serial tree adder is used for various purposes. A global bit-time clock synchronizes the CASSM cells. A CASSM cycle is defined as one revolution of the memory elements or the sum of a scan time and a gap time. The scan time is the amount of time it takes to scan the data on a memory element. The gap time is the amount of time it takes to synchronize with the timing marks that start the memory elements. During the gap time, a certain number of operations such as the aggregate functions

may be performed.

Cell Organization

A processing element in CASSM consists of a number of parallel operating modules. Data and/or program instructions stored on each cell's rotating memory elements flow through a pipeline of modules as shown in Figure 3.

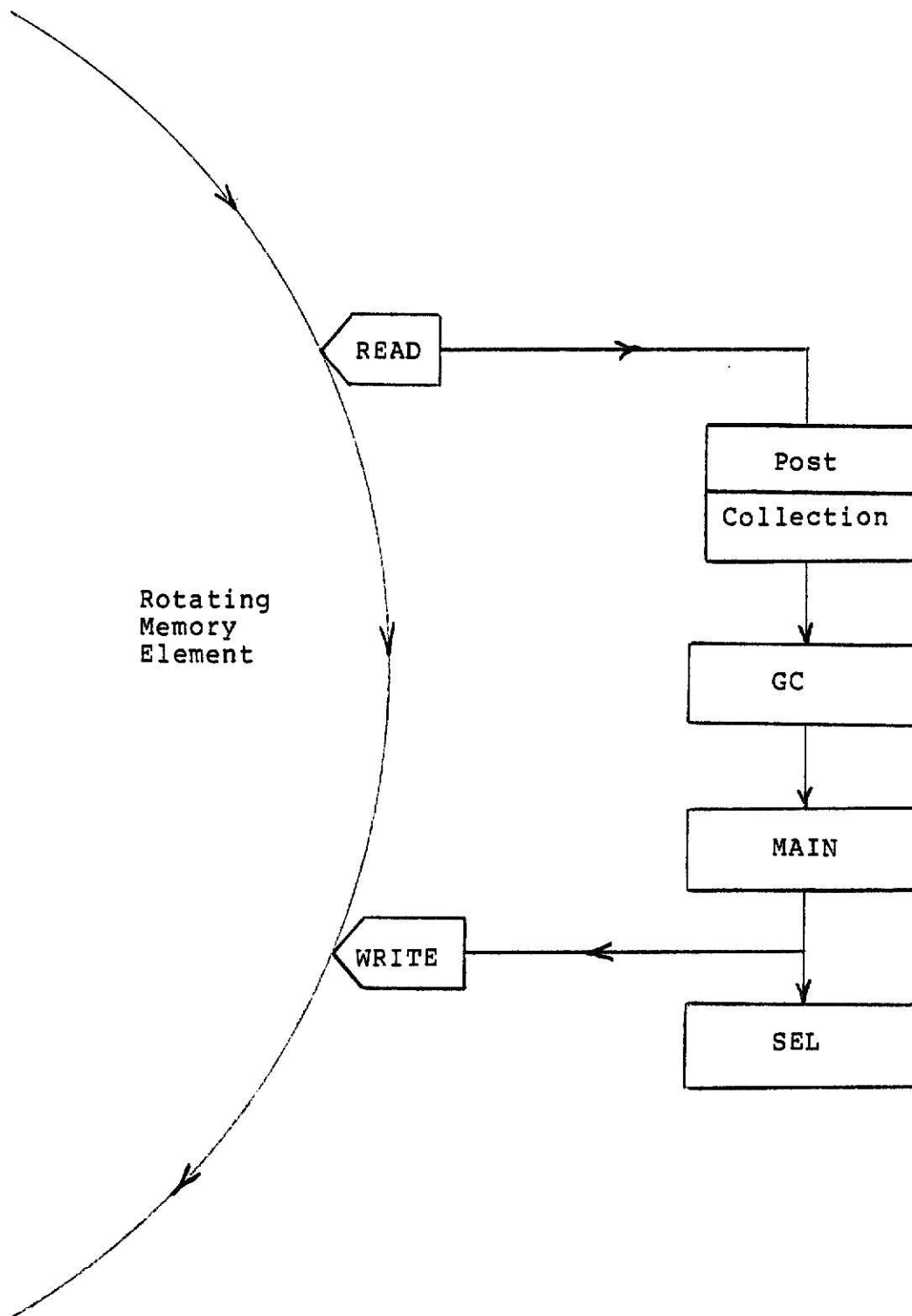


Figure 3. General view of a CASSM cell

The function of the READ head is to pick up each word from the memory, adjust timing (through FIFO), perform parity checking on it, and then pass it on to the POST processor. The previously active instruction and immediate operand which were fetched in this cycle are deactivated in this module. The POST module also finishes the execution of the instruction started during the last cycle. Next the GC (garbage collection) module, which shares part of the POST module's hardware, tries to absorb any words with a tag which says "garbage word". MAIN executes the search part of the instruction which was fetched during the last cycle. WRITE updates words or status bits which are in the proper trees and/or records. Words coming out of WRITE are then written back out on the memory. However, the instruction for the next cycle has to be fetched along with its operands. The context for the searches has to be set up for the next cycle. All this is done in the SEL (selection) module which does not change the data in the pipeline. The same data and instruction words flow into both the SEL and WRITE modules.

Data Organization

The data organization of CASSM is in bit-serial and word-serial fashion. A hierarchically structured file is linearized in top-down left-right order. The data is physically stored in a sequence of 40-bit words in the memory. A 32-bit field is used for data or an instruction, a 3-bit field for status, a 3-bit field for tags, a 1-bit

field for parity, and a 1-bit field for internal use (See figure 4.). The tags are used to mark different data types such as instruction word, name-value pair, delimiter and character string, pointer word, operand, and garbage word. Status bits are used to mark the word for output, to indicate that the word satisfied a match, and for other purposes. The formats of the 32-bit field are different, depending upon the type of word. A special word called a delimiter is used to delimit the files, records, and subrecords in a hierarchy. It contains a level number, a name, a 6 bit stack, a qualification bit Q and a specification bit S. The level number and name are used to name a node of a tree. The Q bit is used to indicate whether the data associated with the node (tree or subtree) qualifies for the next search or not. Thus, selective nodes in the database can be operated upon by the search operation. The S bit is used to specify whether the node is the place for accumulating the result of a search (i.e. whether the node satisfies a search or not).

Some Capabilities of CASSM

1. A known limitation of implementing a system based on a hierarchical model is that access to the sons has to go first through the father nodes in the hierarchy. However, this limitation does not exist in this system. CASSM's processing elements process an

MNEM NAME		T P TAG STATUS						DATA / INSTRUCTION								
	D	DELIMITER			0	0	0	M	H	C	NAME		LEVEL	B-STACK	S	Q
DATA	N	NAME/VALUE			0	0	1	M	H	C	NAME		VALUE			
WORDS	P	POINTER			0	1	0	M	H	C	NAME OR L-POINTER		R-POINTER			
	S	STRING			0	1	1	M	H	C	BYTE	BYTE	BYTE	BYTE		
	I	INSTRUCTION			1	0	0	A	TP							
	ST	STACK/QUEUE			1	0	1	SN	F							
	E	ERASE/TEMP			1	1	1	STATUS								

IDENTIFIER Meaning	
NAME	Code word
VALUE	Code word or binary number
L-POINTER	Code word or record number
R-POINTER	Record number
LEVEL	Binary number
B-STACK	Bit stack for search results
BYTE	ASCII character
TP	Type
	-00 to 10 first word
	-11- immediate operand
SN	Stack/queue number

IDENTIFIER Meaning	
STATUS	Erase status
	- 100 - END OF FILE
	- 110 - HOLD
	- 111 - GARBAGE
	(all other are TEMP)
S	Specification (enables bit stack)
Q	Qualification (enables search)
M	Match (loaded by QSR search result)
H	Hold (controls some I/O)
C	Collect (collect for output)
A	Active instruction
F	Flag (marks end of stack segment)

Figure 4. CASSM word types

instruction simultaneously against every single word stored in the memory elements in every revolution of the memory. In a memory cycle it is as easy to access a son as to access a father in a hierarchy. Nodes in any level can be directly accessed by using their level numbers and names which are explicitly stored.

2. In CASSM architecture, even though many segments may be used to store and concurrently search the database, the different size records are automatically packed together by the hardware. Intersegment communication is provided so that the user need not be aware of where the records are or how many segments are used for each record. This layout has the advantage that the mapping between the information structure as seen by the user and physical structure is greatly simplified. Furthermore, the restrictions (such as one tuple cannot span two memory elements and different relations cannot be in the same memory element) on the physical placement of records and files do not exist in this system.
3. The CASSM hardware is capable of supporting a high-level language for the user.

4. CASSM hardware is capable of supporting the following primitives: searching, insertion, deletion, garbage collection, evaluation of boolean expressions, forward and backward pointer transfer, and the statistical primitives such as MIN, MAX, SUM, COUNT.
5. An important and unique feature of CASSM is that CASSM memory is used for storing programs as well as data. The instruction and operand fetching techniques and the hardware implementation of conventional programming techniques such as loop control, subroutine calls, parameter passing, etc., allow complex programs to be executed directly on the associative memory. This makes an associative system a stand-alone system rather than an adjunct to a conventional processor.

Some Limitations of CASSM

1. The arithmetic computation capability is very restricted. The hardware cannot perform multiplication and division operations.
2. The data type recognized by hardware is very limited. Only character string, pointer, and binary representation of integers are allowed.

3. The speed of output in this system is generally slow since a single channel is used to output data from multiple cells.

These limitations need to be removed and more database management software needs to be developed to make CASSM an efficient database management system. The problem of data integrity, data security, and recovery are presently under investigation.

3.1.4 RAP - Relational Associative Processor

The RAP [19,21] project began in 1975 in the Computer Systems Research Group at the University of Toronto. The system was designed for the relational structured data files, with restrictions on the length of tuples however, the most recent version (RAP.2) [21] reduces some of the restrictions. RAP is an autonomous processor which communicates with an outside general purpose computer (GPC) only to receive its data contents, to receive its compiled programs, and to send back the results of an user's request. RAP can only handle databases smaller than 10^{**8} bytes in size.

Overall Design

An overall configuration of a operational RAP environment is given in Figure 5.

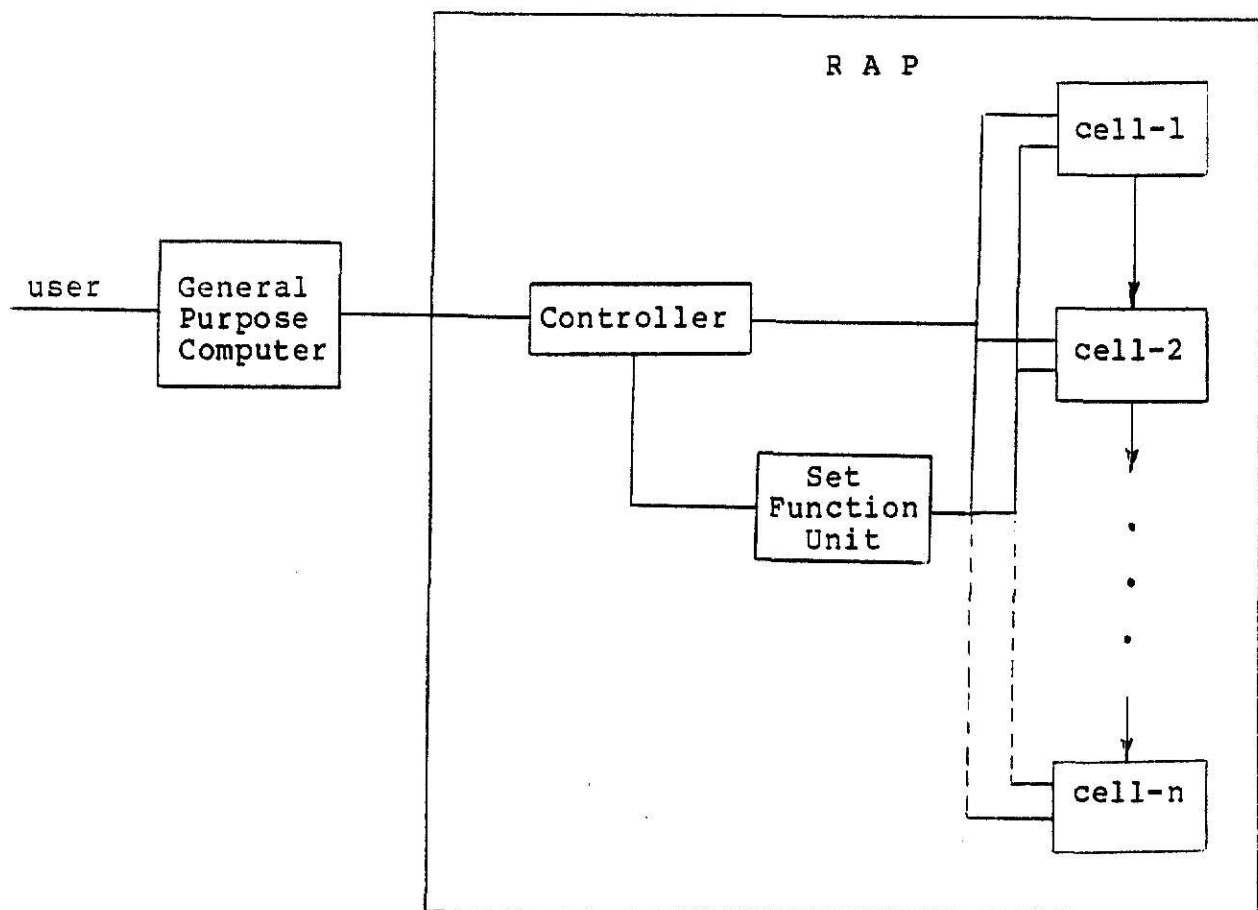


Figure 5. Overview of RAP Architecture

The design is composed of a controller, an arithmetic set function unit, and a parallel organization of cells. A cell consists of a memory component and a logic component (processors). The set function unit is used to combine cell results to obtain a value computed over the total memory contents. The controller is responsible for overall coordination and sends control sequences to the cells, controls the set function unit, and executes decision commands and other RAP primitives that can be accomplished directly in itself.

Cell Organization

Each cell consists of a rotating memory, a buffer, an information search and manipulation unit (ISMU), and an arithmetic logic unit (ALU). The basic logic blocks are displayed in Figure 6.

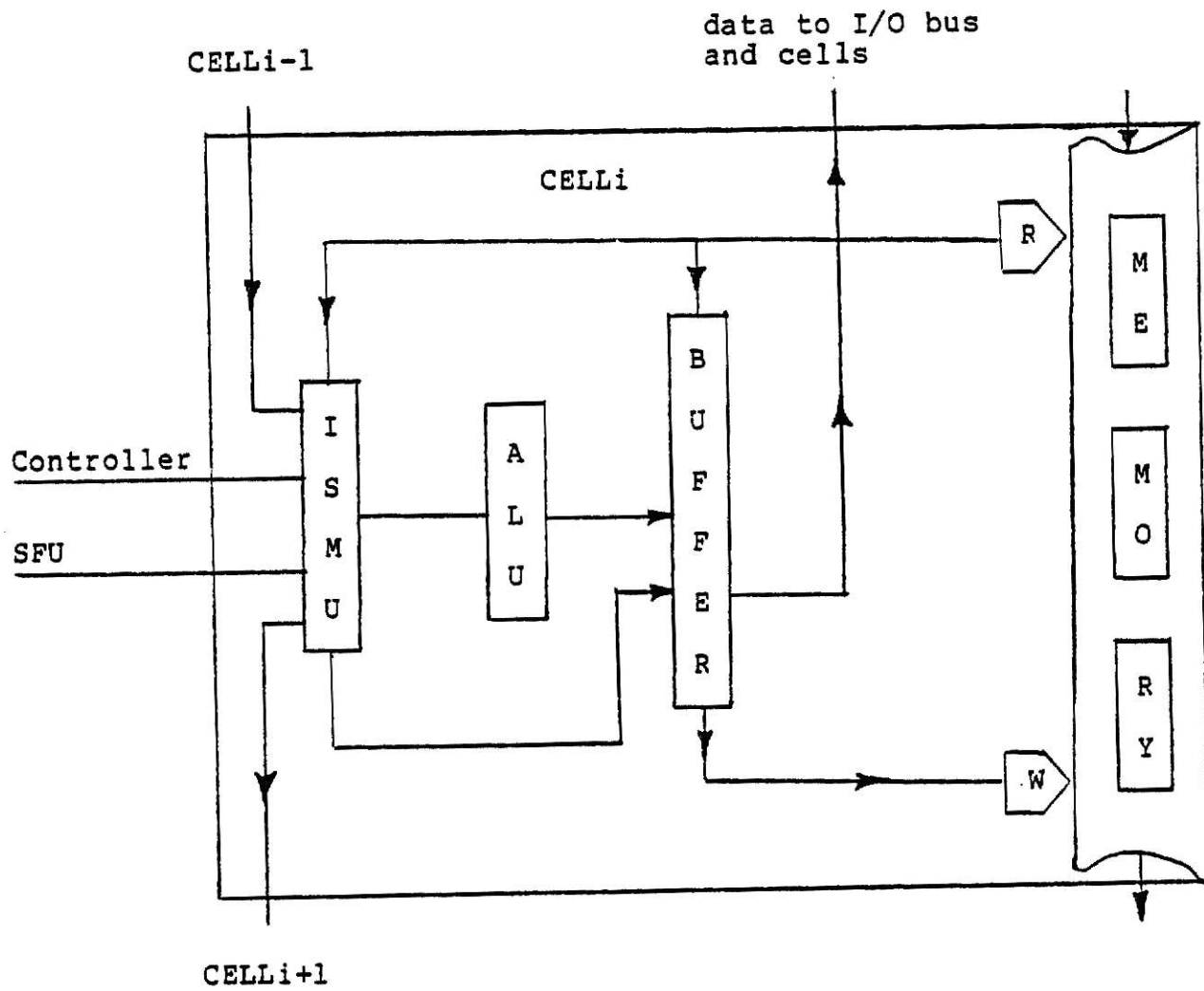


Figure 6. Overview of RAP Cell Architecture

A rotating bulk memory with high track capacity should be selected to achieve low cost per bit of storage. Data is read or written via fixed heads -- one set for each cell -- while the memory rotates under these heads. The associated memory with each cell is called a track. RAP memory space is the sum of the individual cell tracks.

Buffer. As the memory rotates under the heads, it is read, circulated through logic, and written back after a time delay. This delay is proportional to the length of a shift register buffer placed between read and write heads. This buffer has been designed to have a length of 1024 bits which holds a sufficient amount of data exposed to the cell logic to support the logical data structure.

Information search and manipulation unit. This unit is responsible for inter-cell communication, decoding of the commands sent from the controller, evaluation of data search criteria, I/O data transfers, and control of the ALU for data modifications.

Arithmetic logic unit. This unit contains a serial adder, multiplier, control counters, and logic for arithmetic computations and modifications. Logic for intermediate set function calculations (e.g. summation, maximum, etc.) are also present.

Data Organization

The memory associated with each cell is called a track, and due to the linear nature of a memory track, a direct mapping of data structure would require it to be linearized. This is done easily with a relational structure. The tuples of the relation, which are linear representations of domain values, can be stored one after the other on a track (see Figure 7).

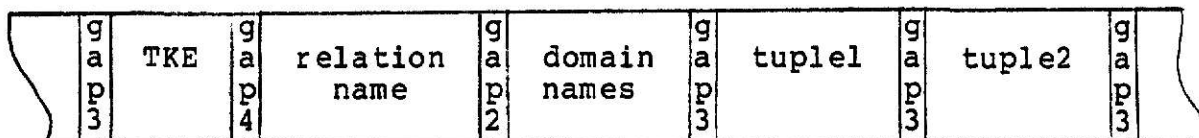


Figure 7. Track Format

The first two data blocks contain relation and domain names respectively and act as "header" blocks. If a relation has too many tuples to be stored on one track, then several cell tracks are used. The relation and domain names, however, should be repeated on each track. A specified length gap is required between every two blocks. The lengths of these gaps are proportional to the amount and speed of logic required between block operations. The beginning of a track is indicated by a marker which is detected electronically. This marker also implies the physical end of a track. The logical

end of a track is indicated by a tuple block which carries a delimiting "track end" (TKE) item.

Instruction Set

The RAP instruction set is designed to construct the basic operations necessary to support databases and, at the same time to be feasible for hardware implementation. These basic functions are:

1. Selection
2. Implicit join
3. projection
4. Free variables
5. Arithmetic set functions
6. Simple arithmetic update.

Selection is applied by applying a boolean search predicate to each tuple of a relation and "marking" or reading the tuples satisfying the predicate. The implicit join operation allows values retrieved from one relation to be used as the retrieval criterion on another relation. The association is made through domains common to both relations. The set operations of union, intersection, complement, and difference can be done on the selected subsets of a relation. Projection is the act of selecting a subset of domains to be retrieved and eliminating duplicate values after a possible selection has occurred. A "free variable" is the term given to an implementation of the following capability. It involves the selection of tuples based upon

the values of domains which occur in another tuples of the same relation

A newer version of RAP called RAP.2 [21] has been designed which is faster, and has a more flexible architecture. There were three important changes in the organization of RAP which resulted in the design and implementation of RAP.2. First, the controller was implemented by a mini/micro computer. Second, the data track was designed around the capabilities of emerging block addressable memories, instead of a disk. Third, a more uniform and flexible instruction set was designed. However, the RAP.2 implementation is still far from perfect and its performance can be greatly improved. For example in RAP.1 and RAP.2, a) each record is limited to 255 items whose length could only be 1, 2, or 4 bytes of encoded data, b) each cell can only store data from one relation, if a relation is large it should be allocated to several cells, c) the execution time is slow when a very large volume of data is to be inserted or retrieved. These limitations can be eliminated by adding more features to the system. The new system which is currently under research is called RAP.3.

3.1.5. RARES - Rotating Associative Relational Store

The RARES [22] project began at the University of Utah in 1976. The aim was providing high performance

content-addressable memory for the realization of a relational database. The RARES hardware operates in conjunction with a query optimizer such as SQUIRAL to support a relational query language. Due to its head-per-track architecture, RARES is also practical only for databases smaller than 10^8 bytes in size.

Overall Design

Physically, RARES is connected to a CPU and buffer memory by a high speed channel. RARES uses a head-per-track rotating disk in which relational tuples (i.e. records) are stored orthogonally across tracks in "bands". A search module is associated with each band to perform access operations on the tuples in the band. The band organization greatly reduces the complexity of sending relational tuples to the CPU for processing. One capability of RARES, which is absent from CASSM and RAP, is its hardware support to produce sorted relations. It can maintain tuples in sorted order or to rapidly sort tuples on domain (i.e. on a record attribute) to facilitate certain kinds of search operations.

Data organization

RARES uses a very different organization from CASSM and RAP. It lays out relation rows across tracks (along the radius of a disk) in byte parallel fashion: the first byte of a value is placed on a track; the second byte of the

value is placed in the same position on the adjacent track and so on. The decision to use a byte-parallel rather than a bit parallel organization was based on the speed of the logic available to process a row laid out along a radius, given the rotation time of the disk. Each set of tracks used to store a relation in this fashion is called a band. The number of tracks in the band may vary; the size of the band is determined by the width of a row. Relations with wide rows may use more than one radius to store a row. This format, called orthogonal layout, is illustrated in Figure 8.

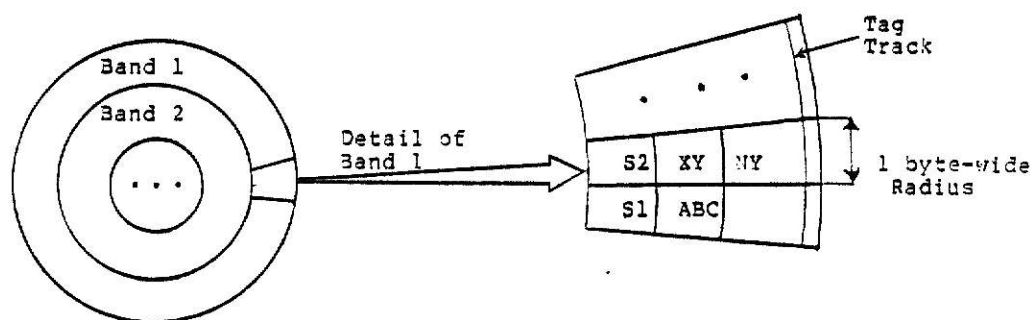


Figure 8. RARES data organization

The ortogonal layout means that fewer rows can come into contention for output. However, some contention is still possible, so RARES also needs an output arbiter. It uses a fast memory, called response store, associated with each band to mark rows to the output on subsequent device revolutions.

3.2. High speed associative memory systems

In these systems [6,7,25,27,28], a high speed associative memory is used together with conventional memory devices such as core memories, rotating memories or shift registers to form a hierarchy of memories for data processing. Databases are stored on conventional secondary storage devices. Data are moved from slower secondary storage to the associative memory for high speed searches by content or context. Figure 9 shows a typical configuration of this type of system.

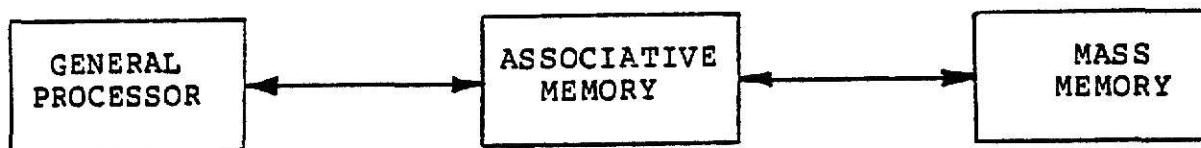


Figure 9. A typical Associative Memory System Configuration

The architecture of associative memory, which is shown in Figure 10, contains four major components: a comparand register, a mask register, an associative array and a set of response registers.

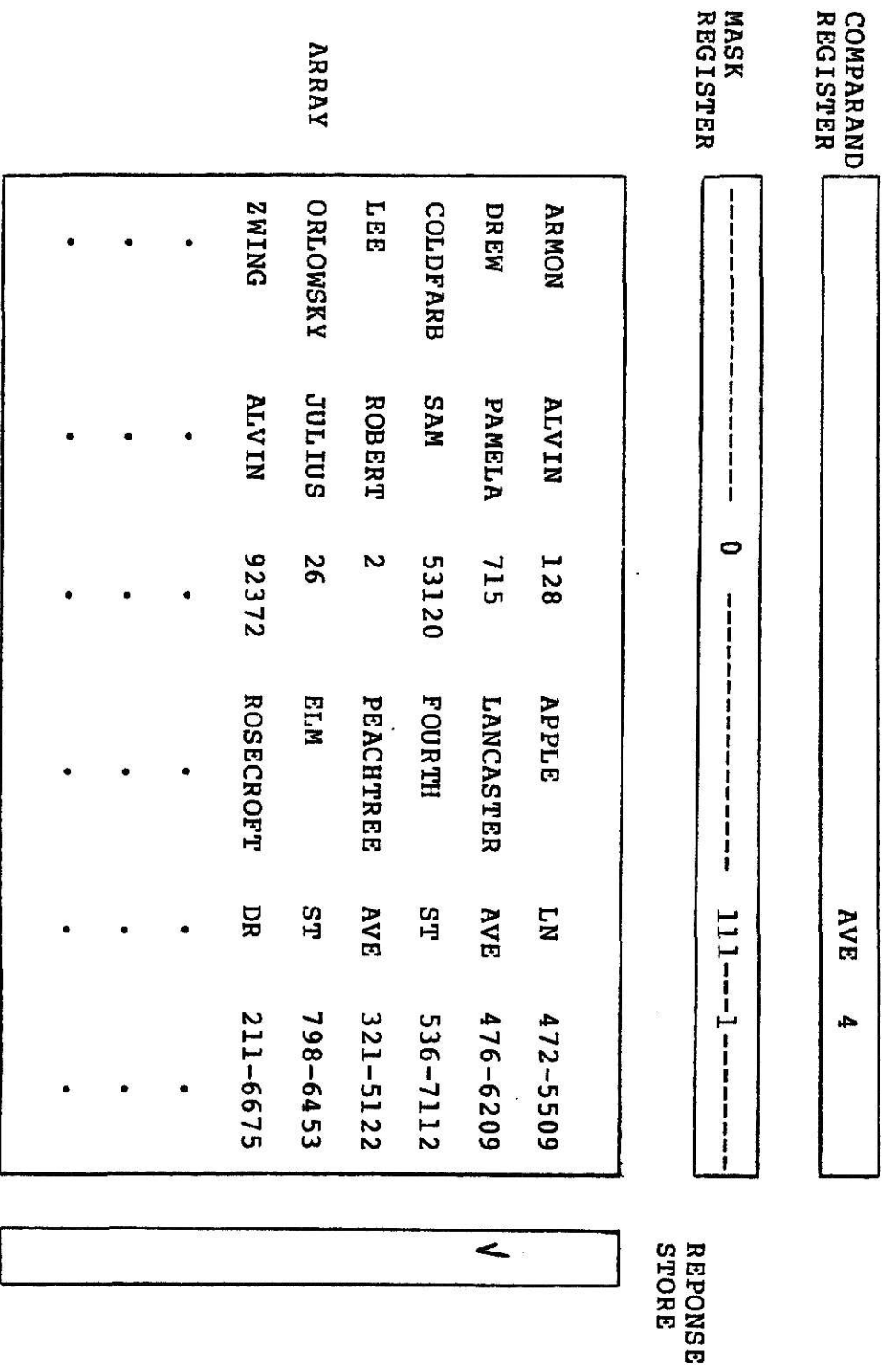


Figure 10. The Architecture of an Associative Memory

In searching a database these four components perform the following functions:

The Comparand Register

The comparand register is used to hold search arguments.

The Mask Register

The mask register is used to indicate which fields (or portion of fields) are to be searched.

The Associative Array

The associative array contains the data to be searched.

The Response Store

The response store record search results and aid in the evaluation of Boolean expression.

Associative processors provide a bit parallel search of the Associative Array. That is, the i (th) bit of each element in the associative array is compared to the i (th) bit of the Comparand Register in parallel. For example, suppose the database is loaded in memory as shown in Figure 10, and we would like to search for the records of those persons who live on AVENUES and have a 4 as the first digit of their telephone number. The Comparand Register is first loaded with AVE and 4 in the proper positions. The Mask Register is then loaded with one in the position of interest and zero otherwise. This has the effect of masking out

unwanted positions in memory. An exact match search is then performed which results in a mark in the Response Store indicating that Pamela Drew's record satisfies the query. The record can then be removed for further processing if required.

3.2.1. Some capabilities of associative devices

The associative processors provide hardware support for the following primitives: addition, subtraction, multiplication, division, content and context searching, evaluation of Boolean operations and statistical primitives such as MIN and MAX.

3.2.2. Some advantages of associative devices

- a. A great advantage of these systems is the rapid search of the associative array since the data can be searched by content and in parallel.
- b. In terms of updating, associative memories also have advantages. Consider a deletion. Once the record to be deleted has been located, the word-masking capability can be used to prevent that record from participating in further operation. Addition can be made to the bottom of any relation since order is not

prerequisite. Changes to a value are easily accomplished using the content addressability property to locate the value to be changed and then writing the new value in its place.

- c. The fact that each bit or any combination of bits of a word can be used as a key for searching indicates that flexibility is increased for associative devices.

3.2.3. Some disadvantages of associative devices

- a. A major disadvantage of the associative devices is the fact that the data to be searched must be moved from secondary storage to the associative array. The time to load data into the array is much greater than the time to search the array. This reduces the performance in a great degree.
- b. The capacity of associative array is currently small (about 250,000 bits) [7]. This requires the frequent staging of data which will result in the above disadvantage.
- c. With the problem of fixed field formatting, as can be seen in Figure 9, the data must be left or right justified in order to exploit the parallel search capability of the memory. This means that the same number of bits must be

allocated to the same data items in each record. This is a waste of storage but must be done in order to allow rapid search.

- d. Associative devices are relatively expensive. However, their capabilities may justify the added expense for certain DBMS applications.

A good example of the high-speed associative memory approach is the STARAN computer system which is discussed in the following section.

3.2.4. STARAN

STARAN [11,20,27,28] computer system was first introduced by the Goodyear Aerospace corporation in May of 1972. Since then it has been installed and is operational at several locations. In 1973, an operational associative processor facility, called RADCAP, was installed at Rome Air Development Center, which consists of a STARAN and various peripheral devices. In 1974 a STARAN was installed by the Defence Mapping Agency (DMA) and the U.S. Army Engineer Topographic Laboratories (USAETL) in Virginia. In 1975 a STARAN was installed at the NASA Johnson Space Center in Houston, Texas.

Goodyear Aerospace Corporation later introduced a new model, STARAN Model E, which is an enhanced version of

STARAN. The following section introduces the system configuration of STARAN and then a few comment are given about STARAN Model E.

System configuration

The basic structure of STARAN is shown in Figure 11. It consists of a number of (up to 32) associative array modules, a control system and a custom interface unit.

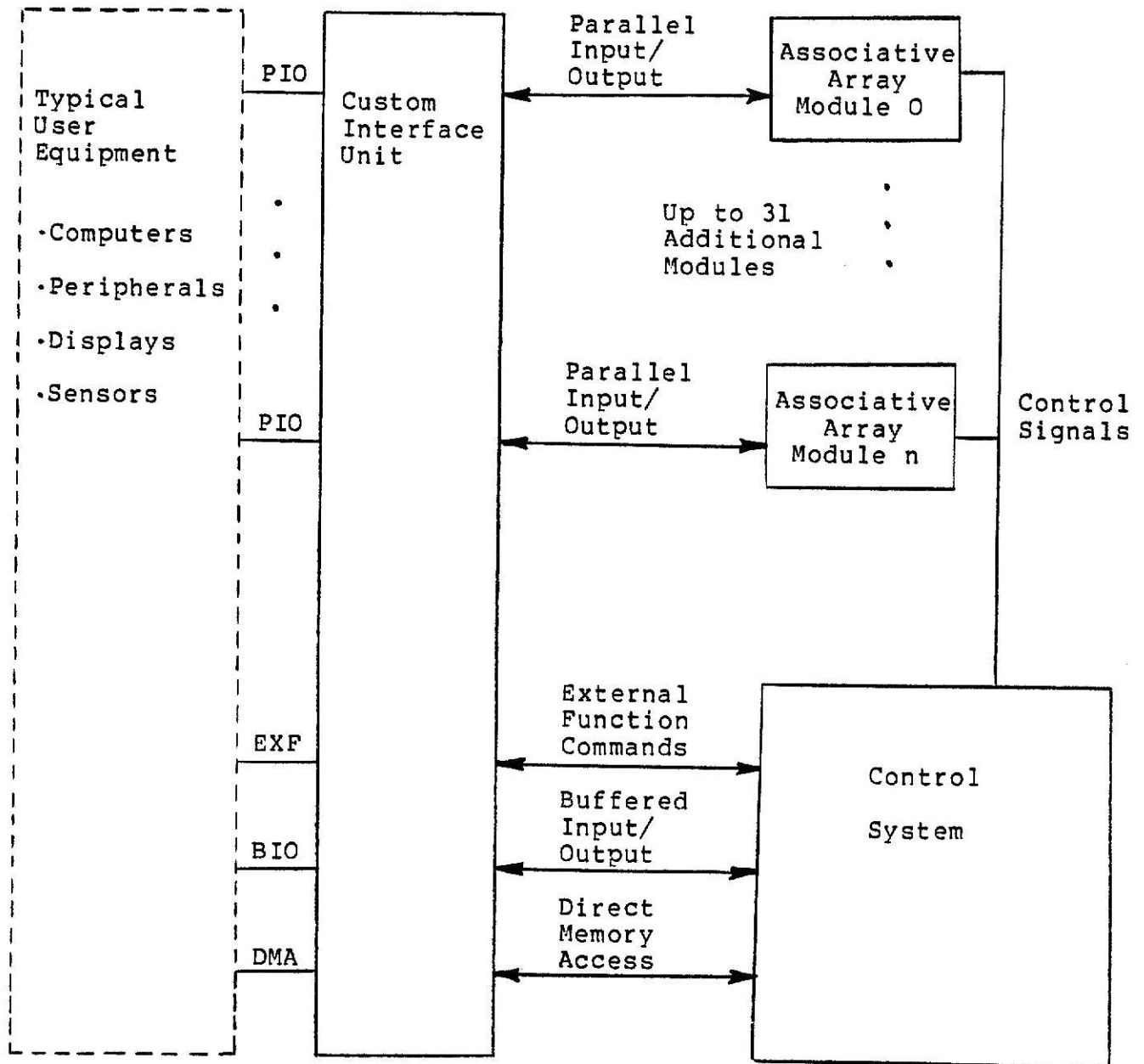


Figure 11. STARAN System Configuration

Associative array module

Each associative array module contains a 256 word by 256 bit multidimensional access memory (MDA). The access can be in parallel in either the word or bit direction. Associated with each word of a processor array is a processing element which examines the content of the word and manipulates the word bit by bit serially. This array of processing elements is often called Response Store (see Figure 12). The unique PIO (Parallel Input/Output) capability is provided by the response store, where every PE has an independent external device I/O path. Control signals generated by the control logic unit are fed to the processing elements in parallel and all processing elements execute the instruction simultaneously. As additional arrays are added to the system these are also connected in parallel to the central logic unit, thus application programs need not be modified as the capacity of the system increases.

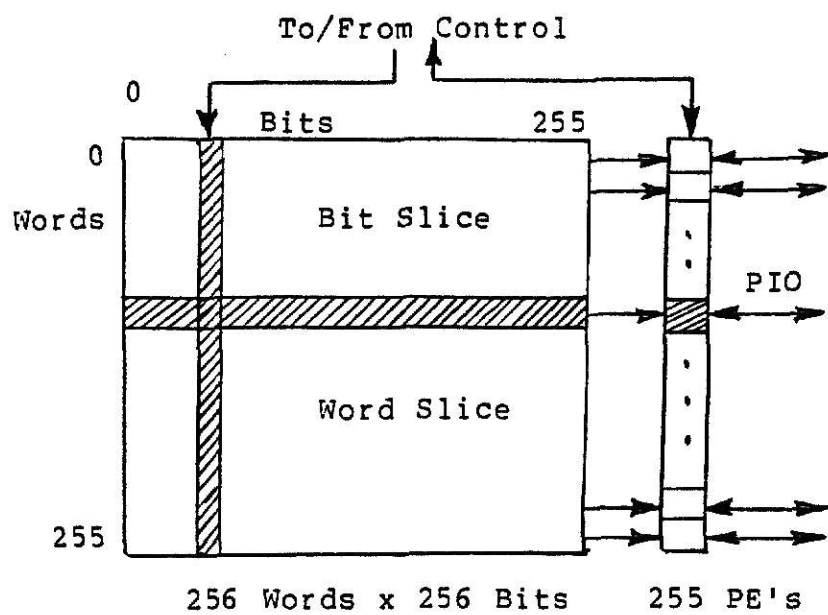


Figure 12. Associative Module

Control system

The major elements of STARAN control system shown in Figure 13 are discussed below:

Associative Processor Control Memory

The Ap control memory is used to store assembled AP application programs and data. Control memory is divided into three fast "page" memories and a slower core memory. The page memories are volatile semiconductor elements, and each contains 512 words (32 bits/word). The core memory uses nonvolatile core storage and it contains 16,384 words. It is used for storing complete AP application programs.

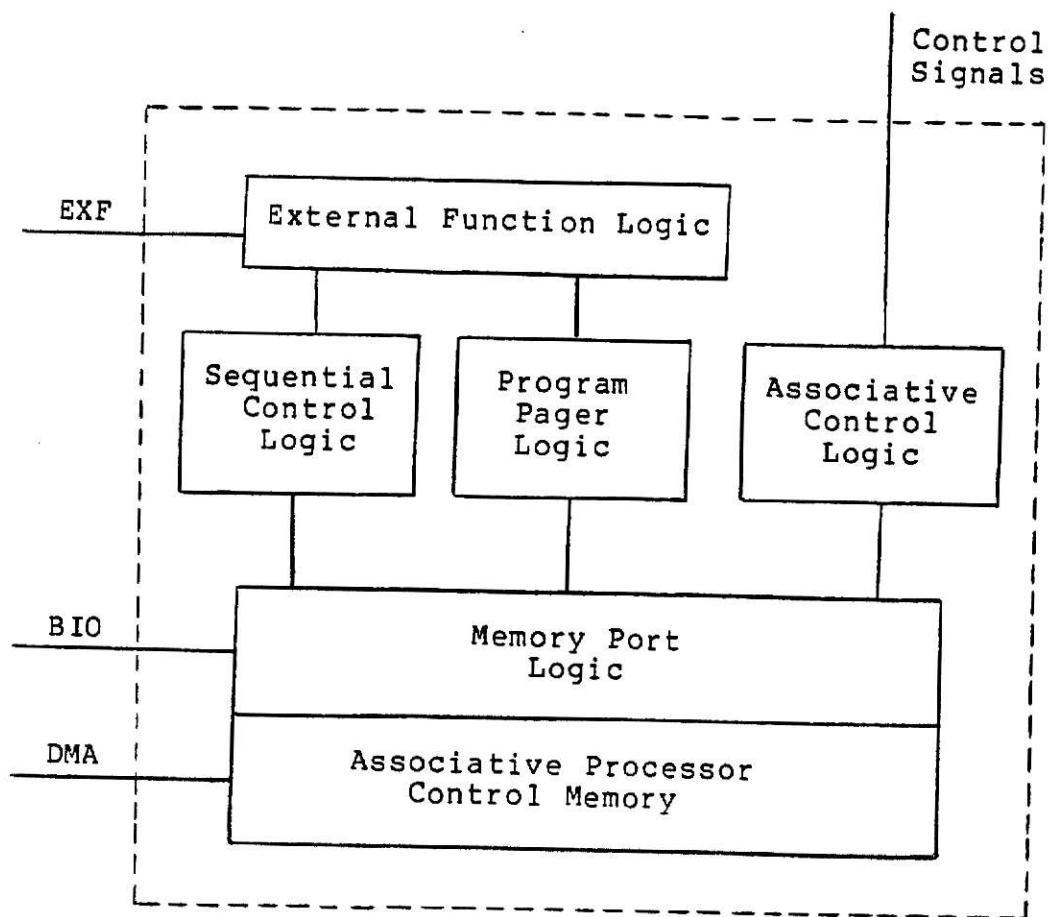


Figure 13. Control System

Associative processor control logic

AP processor logic executes the instructions from control memory and directly manipulates data within associative arrays. It is the data communication path between control memory and the arrays.

Program pager logic

The program pager loads the fast page memories with the data from the slow core memory. While the AP control is executing a program segment out of one page, the pager can be loading the other page with a future program segment.

External function logic

External function logic enables the AP control, sequential control or an external device to control the STARAN operation. By issuing external function codes to EXF a STARAN element can interrogate and control the status of the other elements.

Sequential control processor

The sequential control (SC) consists of a sequential processor, a keyboard-printer, a perforated type reader/punch unit and logic capability to interface the sequential processor with other STARAN elements. SC is used for system software programs such as assembler, operating system, diagnostic programs, debugging and housekeeping routines.

Custom interface unit

The interface unit provides communication to sensors, conventional computers, signal processors, interactive display and mass storage devices. A variety of I/O options are implemented in the custom interface unit, including the direct memory access (DMA), buffered I/O (BIO) channels, external function (EXF) channels and parallel I/O (PIO). Each associative array module can have up to 256 inputs and 256 outputs into the custom interface unit. They can be used to increase speed of inter-array data communication to allow STARAN to communicate with a high-bandwidth I/O device, and to allow any device to communicate directly with the associative array modules.

Software

STARAN software system consists of an assembler language called APPLE [11] (for Associative Processor Programming Language E), and a set of supervisor, utility, debug, diagnostic and subroutine library program packages and a disk operating system (DOS) which has a batch processing capability.

APPLE stands at a higher level than sequential machine assembly languages because of the capability of parallel search and parallel arithmetic operation.

STARAN Model E

Although the original STARAN had satisfied system

performance predictions for a variety of applications, for large applications it became slow and insufficient. Therefore the enhancement of the original system became necessary and the new technology and improvement in the industry made this enhancement easier. The STARAN architecture was therefore enhanced as follows [4]:

1. Faster multidimensional access (MDA) arrays that provided a minimum of 36 times the storage capacity of the original STARAN arrays.
2. New hardware that allowed inter-array data transfers from 8 to 64 times faster.
3. New page memories that provided 8 times the storage and allowed up to 65% faster array instruction execution times.
4. A set of floating point arithmetic modules that allowed the STARAN programmer to arbitrarily specify the mantissa and exponent lengths.

3.3. Backend computers

The basic idea behind the back-end concept [9,10,18,25] is shown in the Figure 14.

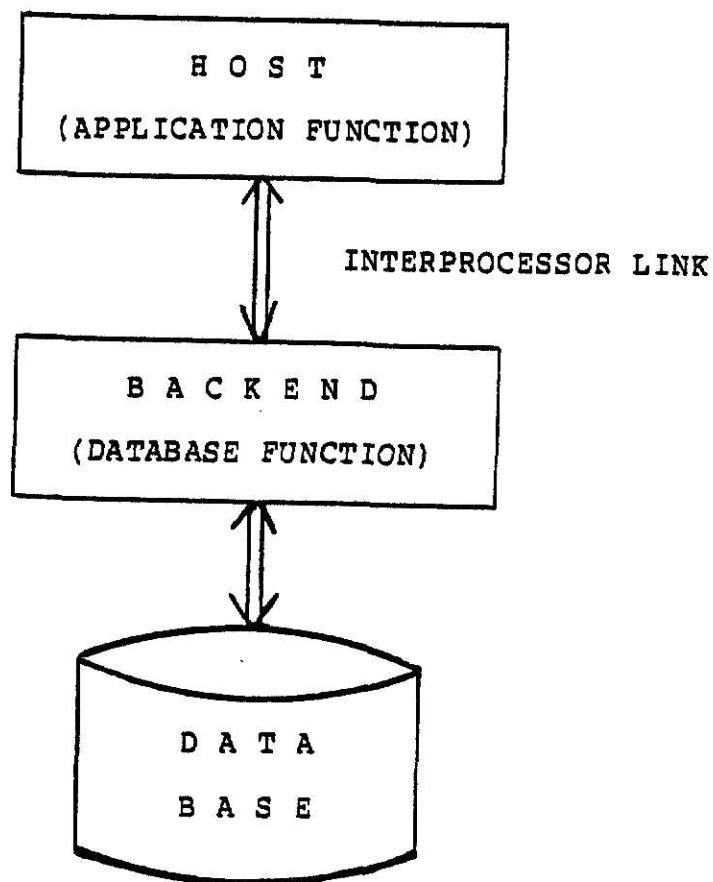


Figure 14. A Configuration of a Backend Computer System

The back-end computer is used to carry out database management functions separately and off-load these functions from the host computer. The back-end computer may be a general purpose computer or it can be a specialized hardware device capable of handling DBMS functions. In either case, it will interface with the host computer. This interface will be responsible for collecting the database management requests from the application programs and transmitting them to the back-end. In turn, it will accept results and status from the back-end and distribute them to the application programs.

There are several advantages and some disadvantages to this approach which we will discuss in the following section.

3.3.1. Advantages

1. Release of the host computer from tedious and time-consuming operations involved in database manipulation, maintenance , and control.
2. Increase of the system performance through functional specialization and through parallel processing among the host and the backend(s).
3. An enhanced ability to share data between computer systems. As shown in Figure 15, a single backend can handle the processing of

the database and present data in forms
suitable to dissimilar host.

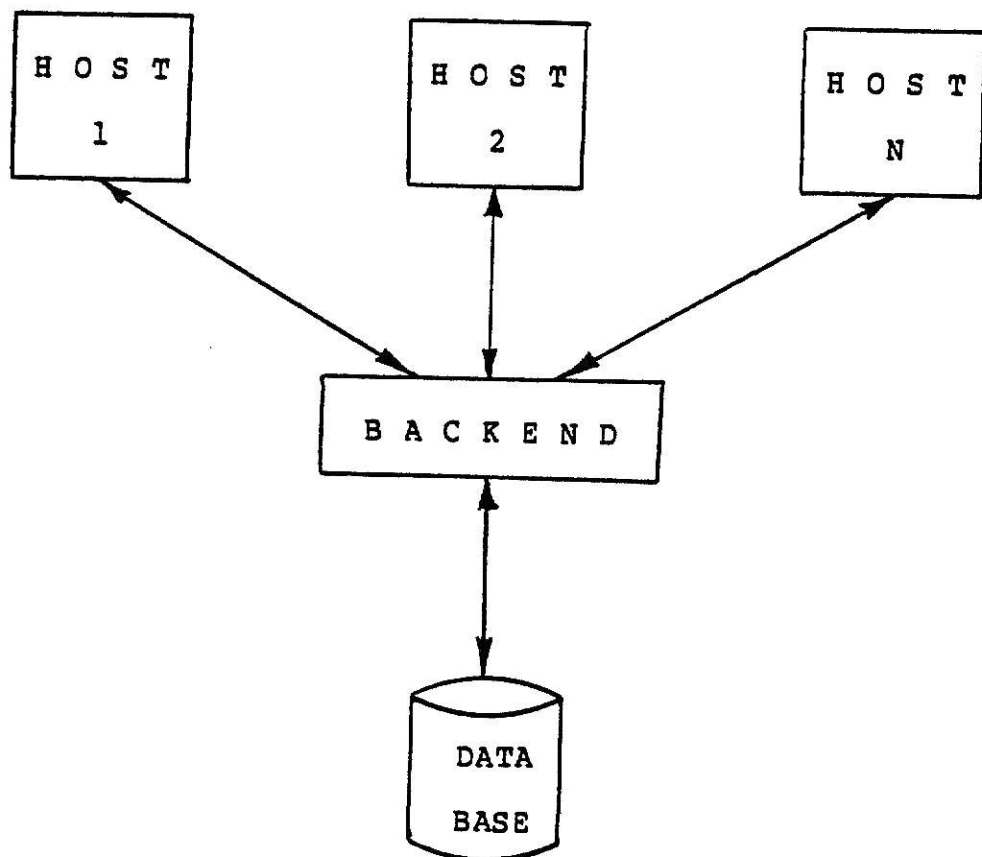


Figure 15. Multiple Host Configuration

4. Changes to the database, the mass storage devices and DBMS will not require changes to the host.
5. Multiple numbers of backends can be used to process large databases. (see Figure 16). These databases can be stored either in a distributed manner across secondary memory devices to facilitate parallel processing, or in a manner such that one database can be processed by one backend.

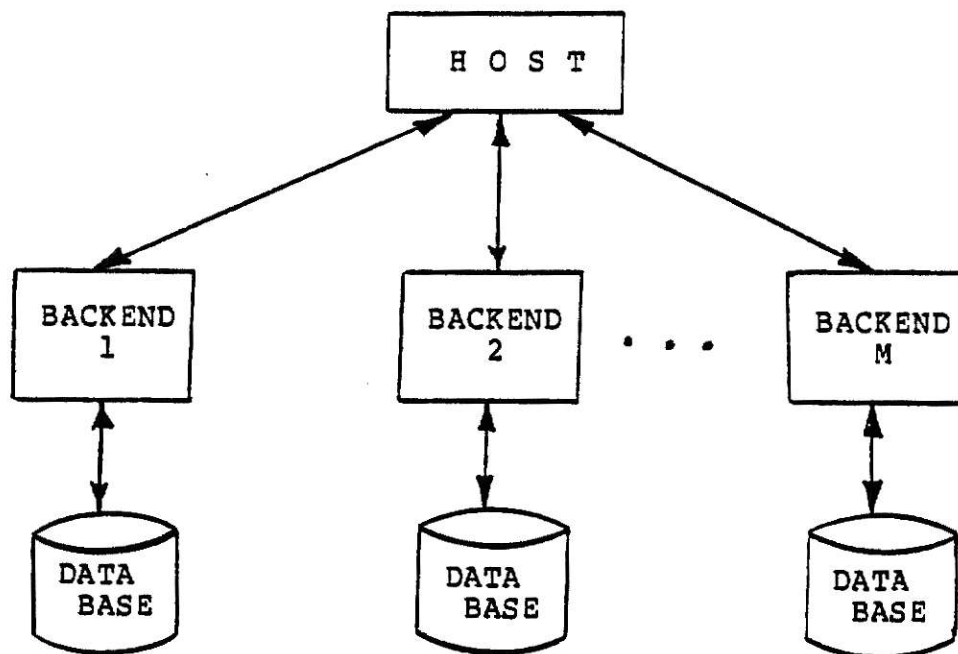


Figure 16. Multiple Backend Configuration

6. Storage devices including special purpose cellular-logic devices or bubble devices can be made available through backends to mainframes that do not otherwise support these devices because of I/O or operating system constraints.
7. The reliability of a DBMS is enhanced, because the inter-machine communication permits the machines to check the information passed between them and verify each other's operation. The machine errors can therefore be detected more quickly and reduce the length of time in which the system operates with an undetected error.
8. A higher level of security is achieved, simply because an application program can not access directly any file, the access can occur only through the DBMS. The unauthorized user therefore will have more difficulties in accessing data.
9. A back-end computer is economical because its acquisition cost is an order of magnitude cheaper than replacing the mainframe with a larger machine.

3.3.2. Disadvantages

1. Since there are more components in a backend DBMS composed of two machines than there are in a single machine system, the incidence of failure of either software or hardware, is higher.
2. The response time decreases because of the transmission time required for command to communicate between the back-end and the host.
3. The problems associated with contracts, maintenance procedures, operator training, systems programming support, etc. may become duplicated if the backend and host are manufactured by two different vendors.

An analysis of the potential advantages and disadvantages of the back-end concept led to the conclusion that the approach held promise. In November 1970 therefore, development of a prototype back-end system started. This prototype backend system which is called the Experimental Data Management System (XDMS) will be discussed in the following section along with some other systems.

3.3.3. XDMS - Experimental Data Management System

The XDAMS [9,18] project started at the Bell

Laboratories in 1970. The primary purpose behind the implementation of XDMS was to demonstrate the feasibility of the back-end concept. From this point of view, XDMS was a success. The Data Base Task Group (DBTG) language was selected for this project which not only provided the back-end system with an advanced data management language but also provided the DBTG effort with valuable implementation feedback.

System configuration

The overall system configuration is shown in Figure 17. In this system a UNIVAC 1108 is used as the host computer. This is because the DBMS-1100 database management system, which is a CODASYL-based package, already existed on the UNIVAC 1108. The META-4 machine was used as a backend machine. A key factor in the selection of META-4 was that META-4 is microprogrammable, and the designers initially felt that several primitive database functions would have to be implemented in microcode to achieve adequate performance.

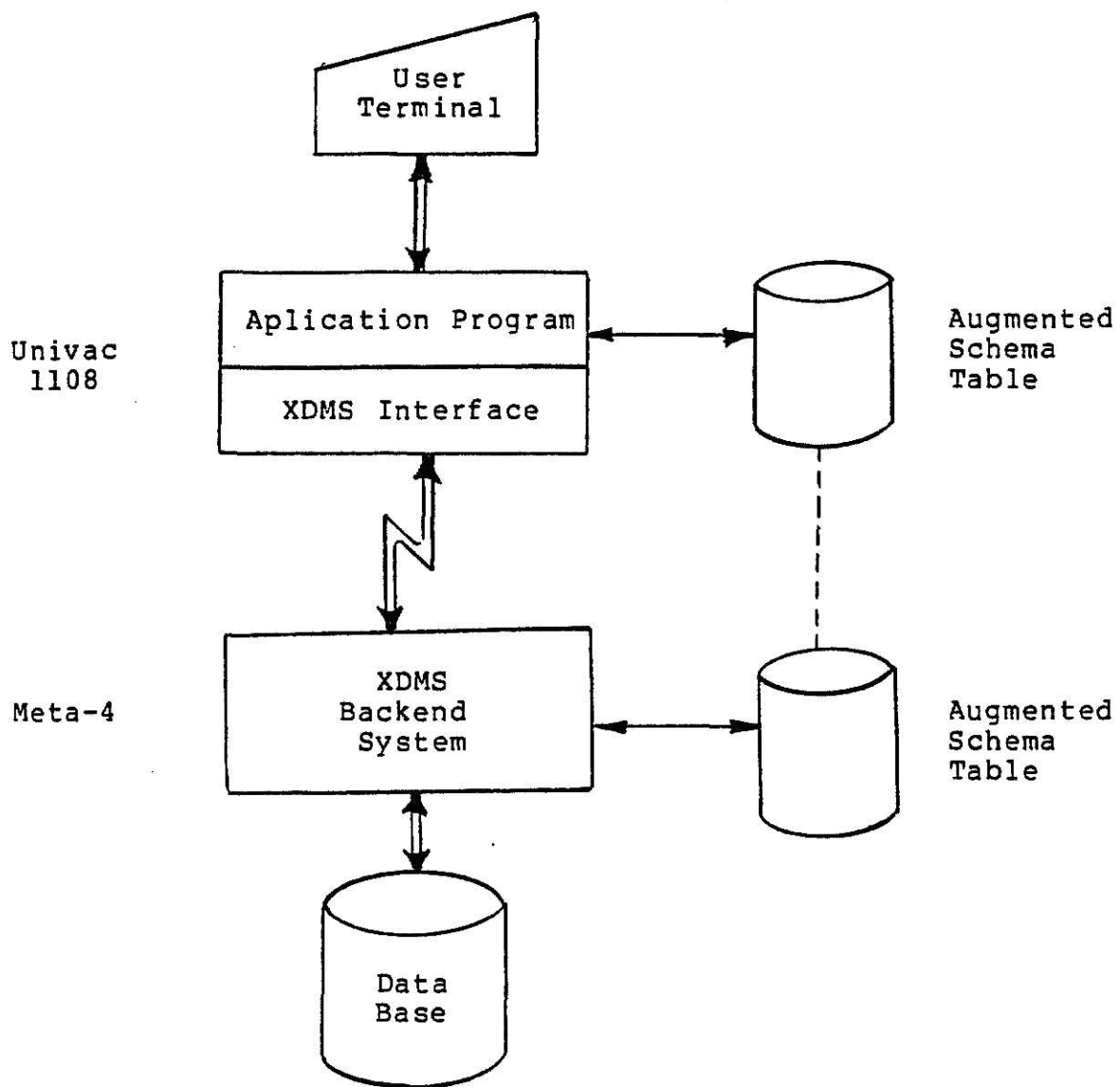


Figure 17. XCMS Hardware Configuration

There has been several other factors which led to selection of this machine, such as inexpensive cost of this machine, the availability of IBM 1130 software through emulation on META-4 and the fact that it has interleaved memory with I/O cycle-stealing, which allows the rapid transfers of large amount of data to and from the core memory.

As shown in figure 7, The schema tables are augmented to both host and the back-end to include some additional DBTG features not get handled by the UNIVAC DDL compiler. The UNIVAC execution time system has been replaced by the XDMS interface, which controls the communication between the UNIVAC 1108 and META-4.

A request typed at a user terminal is first passed to the application program for syntactic and semantic analysis, having interpreted the request, the application program would then issue the necessary DML command(s) to the XDMS interface program. The command would then be encoded and transmitted to the back-end system over the data link. The XDMS system in the back-end would interpret the DML command and would access the database using schema table information. Having executed the command, the back-end would transmit the results back to the interface program in the host, the result is then passed to the application program for display at the user terminal. A request by a batch run is essentially the same. Any number of user request languages could be developed for the system by design of the

suitable application programs. In particular, an interactive database management language, called DATABASIC, which allows on-line manipulation of a database, was developed as a part of XDMS.

Some capabilities

1. Multi-user system. XDMS was designed to handle many users simultaneously. In the host computer, this was accomplished by designing the XDMS interface so that commands can be collected from, and result distributed to, a number of application programs.
2. Locking mechanism. XDMS system provides a locking mechanism to control concurrent updates. The locking can be done on a physical record within a page or on fields within a record. Actually in XDMS a logical DBTG record is separated into two physical records (pointers and data) so locking an entities within a logical record is provided. Associated with the locking problem is the deadlock problem. The XDMS system provides the capability for one user to back off once deadlock is detected.
3. Rollback and recovery. XDMS provides rollback

and recovery features. Actually rollback exists at two levels: command and transaction. Command rollback allows the effects of a given DML command to be erased if a problem is encountered in the back-end. Status is then returned to the host indicating that the command was not executed. Transaction rollback allows the effects of a series of DML commands (a transaction) to be erased.

In conclusion the modular system organization and offloading of host resources to the backend are significant achievements of the XDMS project. The XDMS project is a landmark in the database management area, since it introduced an alternative database architecture and spawned an enormous amount of follow-on research.

3.3.4. DC - Data Computer

The Data Computer [15,17] is another example of the back-end processor approach. It is a large-scale database management system running on a PDP-10 and has been implemented for use in ARPANET in 1973 by Computer Corporation of America. The Data Computer essentially provides facilities for data sharing of a single database

among dissimilar host computers in a network environment.

Logical design

Logically, the Data Computer system can be viewed as a box which is shared by a variety of external processors. The communication between the Data Computer and processors are done by a so called "datalanguage". See Figure 18.

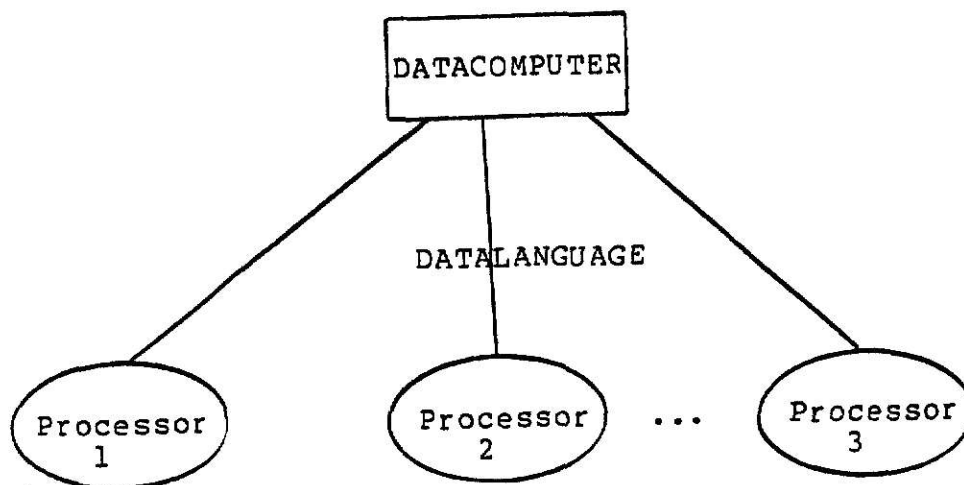


Figure 18. Logical View of Data Computer

A database stored on the Data Computer is sharable by all computers having access to the system. Character codes, floating point number representations, and word sizes vary from user to user; so do the representations of variable length and variable structure, as well as high level data structure attributes. The Data Computer system is required to perform translations between various hardware representations and data structuring concepts. Characters, bytes, and numbers are stored under the control of the machine storing the data. The machine reading the data specifies the format it requires. As data is output, the indicated data conversions are performed.

Datalanguage

Data language is the language in which all requests to the Data Computer are stated. Datalanguage includes facilities for data description, for database creation and maintenance, for selective retrieval of data, and for access to a variety of auxiliary facilities and services.

Physical design

The architecture of the system is shown in Figure 19. The system processor is a DEC System 10 (PDP-10). Memory is present at three levels: primary (core), secondary (disc) and tertiary (mass storage). Peripherals are used for software development and for input of data from tape. The system is interfaced to the Arpanet IMP (Interfaced Message

Processor).

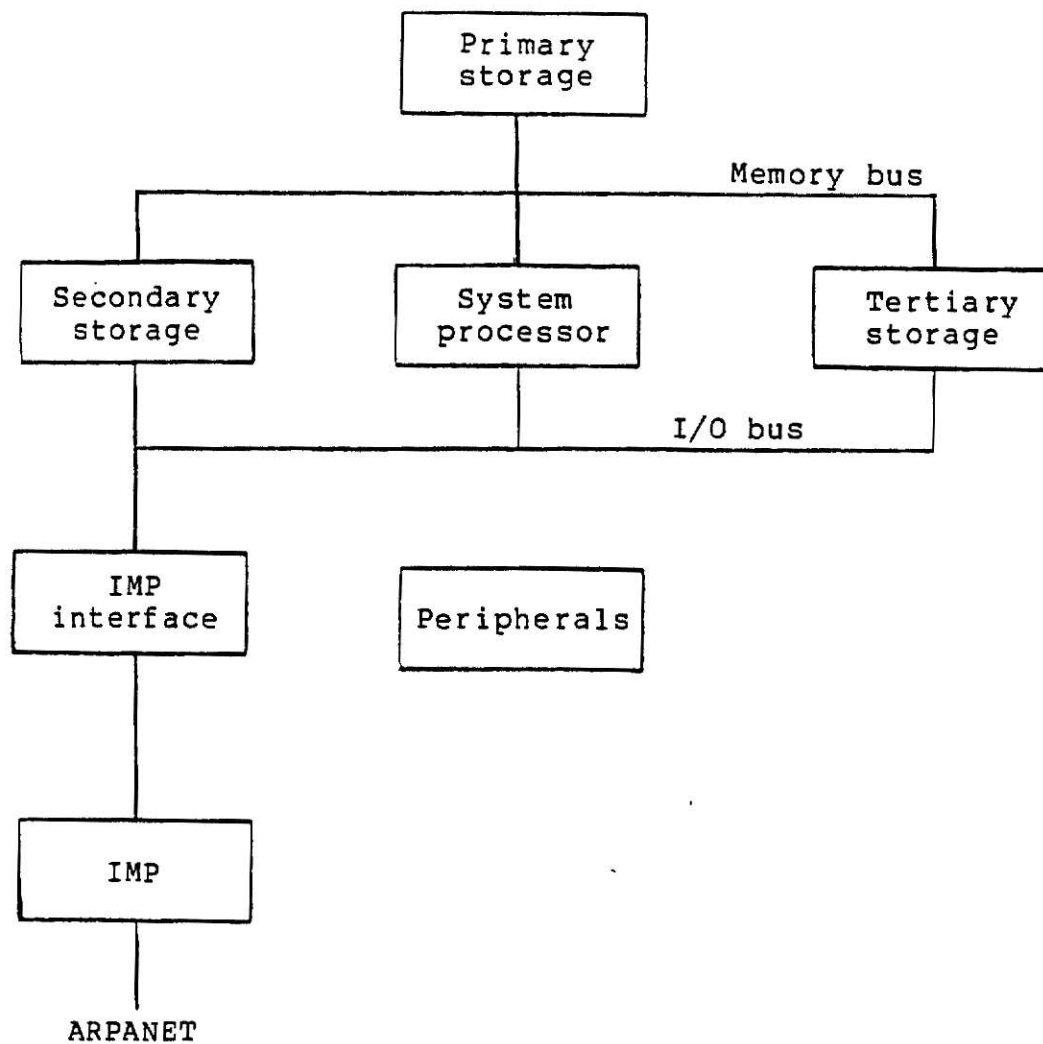


Figure 19. Hardware Overview of system

Data organization

The data is physically organized into pages which move among the three levels of storage: primary, secondary and tertiary. The movement of pages is dictated by various staging strategies. The particular strategy used is selected by the system to optimize the requests currently being executed. Some examples of staging strategies are as follows:

- a. move the whole file to disk and work from disk. This strategy is suitable for small files which can easily fit into the secondary storage buffer area.
- b. Move pages from tertiary store to primary, process the pages, and output directly from primary. This is used when, for example, only a small portion of the data read from tertiary storage to be sent to the user.
- c. Break the request down so as to operate on a segment of the file, and stage to disk one segment at a time. This is useful when particular information is available (e.g. from inversion tables) that indicates those segments which are not needed and can be skipped.

We should also indicate here, that the file system is designed to provide inversion tables for calculating the

location of data in storage.

3.3.5. IDM _ Intelligent Database Machine

IDM [13,14] was developed by the Britton-Lee, Inc., in September of 1980. The system is used to handle relational database management tasks. The system can operate as a back-end machine or, together with intelligent terminals, as a standalone data management system. It is specially designed to support very high transaction rates and process complex transactions that span an entire database. IDM features parallel and pipeline processors and it can handle multiple databases containing up to 32 billion bytes, using moving head disks.

The application program running on host computer communicates with the IDM using a high level nonprocedural command language. The command language allows for defining a database, specifying indices on relations and doing retrieval, update and aggregation.

System configuration

The overall configuration of IDM is shown in Figure 20. It consists of up to eight input/output channels, up to 4 disk controllers, up to 12 random access memories, a database accelerator, and a database processor.

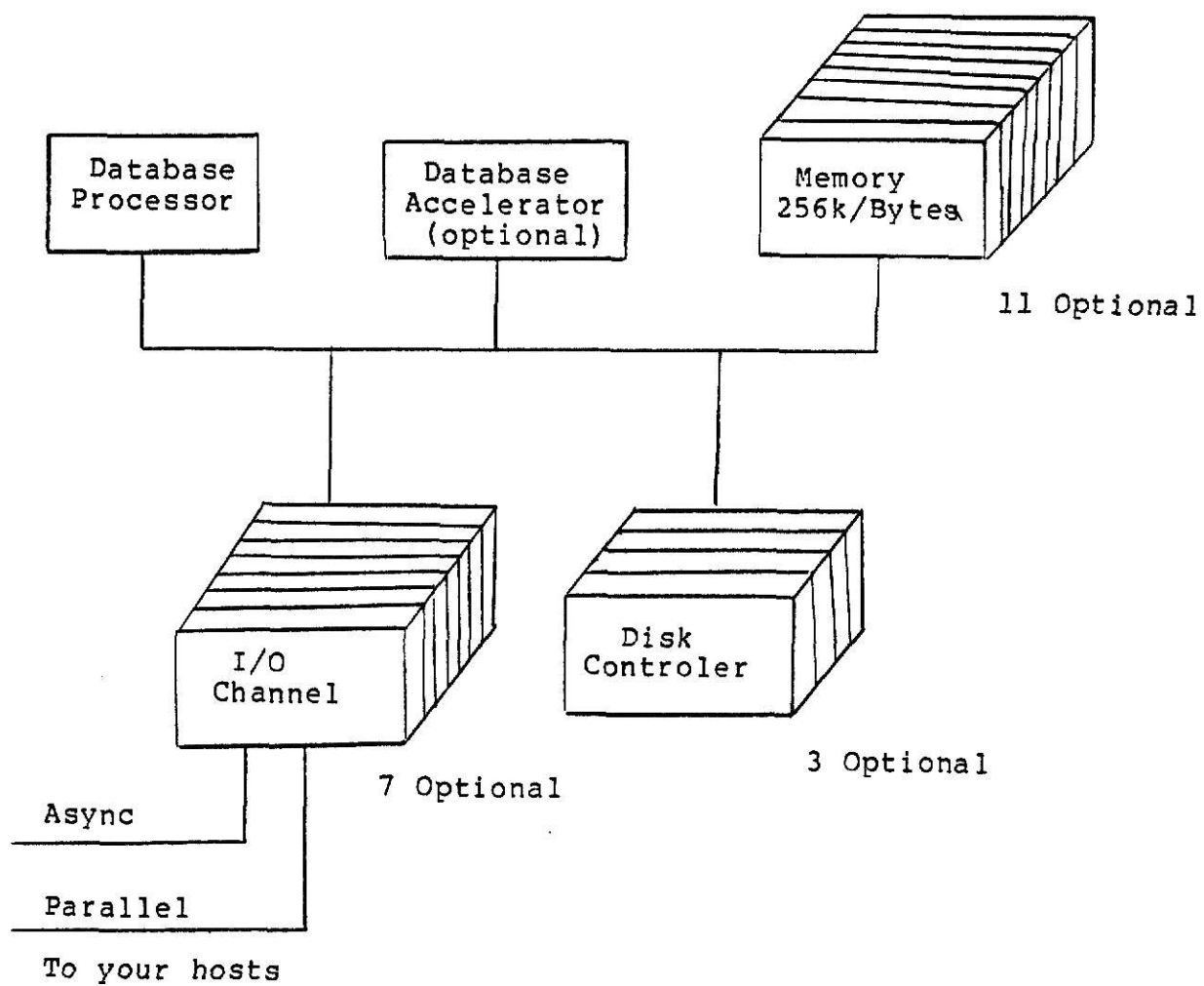


Figure 20. IDM Configuration

Input/output channels

IDM has eight input/outputs that can accomodate up to eight programmable terminals using asynchronous communications. The communication with the host can either be bit serial or byte parallel.

Disk controllers

The disk controllers can support up to 4 disk drives. The disk scheduling is done by the controller. By disk scheduling the controller can allow multiple transactions at a time. When an operation is complete the controller will respond to the host that that operation was completed.

Random access memories

Up to 12 random access memories (256 Kbytes each) can be used to increase the access time. The data is moved from the disks (secondary storage) to these memories for faster execution.

Database accelerator

The database accelerator increases the speed of the machine up to ten times faster, achieving operating speeds of up to 10 million instructions per second. The accelerator takes care of searching, comparing data, matching, and responding to the mainframe CPU or intelligent terminals. When only a small transaction rate is required, the database accelerator can be removed, reducing the cost

of the machine.

Database processor

The database processor is a microprocessor for processing all the database management functions

Other capabilities

Other capabilities of IDM included:

1. System integrity, including protection from power failure, disk failure and other common system problems.
2. An integrated data dictionary.
3. Constraints to limit access to relations, portions of relations, specific records.
4. A view mechanism to allow alternate definitions of relations or collections of relations.
5. Use of indexing/hashing techniques for searching.

3.4. Integrated database machines

This category of systems [25] uses a number of functionally specialized processors, which can be general purpose and/or special purpose processors. For example, systems of this type may use, specialized associative

processors for the processing of directories and mapping data, intelligently controlled disks and mass storage devices for the storage and processing of the major portions of the database, a system processor for general coordination, and dedicated hardware for security control.

3.4.1. Advantages

1. A greater efficiency is achieved by the use of functionally specialized hardware.
2. The modular family of machines allow users to exploit parallel processing and pipelining techniques.
3. These system are more complete and capable of handling large-scale databases.

3.4.2. Disadvantages

Since the system uses dissimilar hardware, the hardware interconnection, the data and program communication and the operating system support is rather complex.

In the following section two systems, the Data Base Computer (DBC) and DIRECT which fall into this category, are introduced.

3.4.3. DBC - Data Base Computer

The Data Base Computer (DBC) [2,3,8,16] was designed at Ohio State University. It was designed to act as a back-end machine to one or more front-end general purpose computers, and support large scale databases (10**10 bytes in size). However, it differs from back-end systems in such a way that it does not require staging of data between levels of memories of various speeds. Further advantages of the DBC accrue from high-level query languages for interface with the front-end computers, and from a content-based security mechanism for access control. The system can support relational, network, hierarchical, and attribute based models.

Overall architecture

The DBC contains seven functionally specialized components. The database command and control processor (DBCCP), the security filter processor (SFP), the mass memory (MM), the structure memory (SM), the structure memory information processor (SMIP), the index translation unit (IXU), and the keyword transformation unit (KXU). These components are organized into two loops, the structure loop, and the data loop (see Figure 21). All the front-end computer systems which communicate with the DBC are called "program execution system" (PES) in this figure. User programs reside in the PES, and are executed by the PES

using the DBC as one of its various resources.

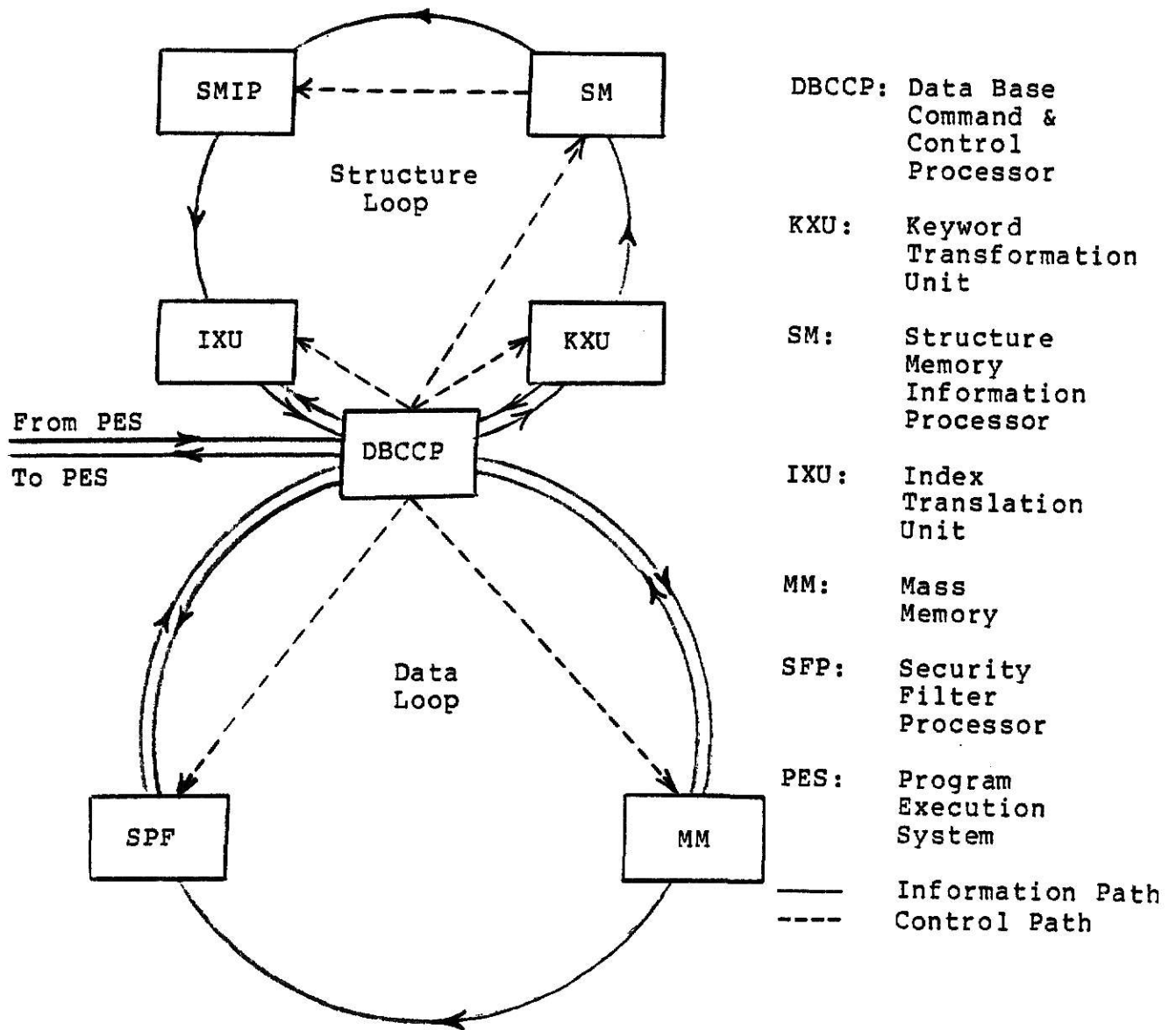


Figure 21. The Architecture of DBC

Data Base Command and Control Processor (DBCCP)

The DBCCP synchronizes and controls all the DBC components. It regulates the operation of both structure and data loops and interfaces with the front-end computer systems, consequently the components can work concurrently on one or more commands. The DBCCP processes all DBC commands retrieved from the front-end computer system, schedules the execution of the commands on the basis of the command type and priority, enforces security on a selective basis, clusters records to be stored in the DBC, and routes the response data to the front-end computer systems. The variable length commands are sent to the DBCCP by the front-end program execution system (PES); appropriate responses, such as set of records, diagnostic messages, etc. are then sent back to the PES by DBCCP. Other functions of the DBCCP include coordinating the task of security checking during database accesses, instructing the SFP to post-check the response set for the field-level control and performing certain essential bookkeeping.

Security File Processor (SFP)

The SFP is used to enforce the field level security of the database. After the records have been retrieved from the query, they are individually checked for security clearance by the SFP. The input to the SFP consists of records retrieved from the MM, and commands and security specification from the DBCCP.

SFP also provides a sorting mechanism. This mechanism enables the response data to be ordered by values of certain attributes. This is usually the way that the user application programs would like to receive the records in the front-end computer system.

Input to the SFP consists of records retrieved from the MM and commands and security specifications from the DBCCP.

Structure Memory Information Processor (SMIP)

SMIP converts the logical identifier from SM into physical disk address.

Mass Memory (MM)

The on-line mass memory (MM) consists of moving-head disks. The disks are modified to allow parallel read-out of an entire cylinder in one revolution time, instead of one track a time. The design of MM is based on the PCAM (Partitioned Content Addressed Medium) [2] concept. Each cylinder of the disk represents one PCAM partition.

The cylinder is made content addressable by incorporating track information processors (TIPs) (one for each track of a cylinder) for concurrent processing of the tracks of a cylinder. Furthermore, the disk read/write mechanism is modified to allow parallel read/write of all the track of a cylinder. By far the most powerful operation of the MM is the search and retrieving records which satisfy queries made up of keyword predicates.

Structure Memory (SM)

The SM is the repository of the directories of the files. The information defining the directories of the files are stored as a set of tuples called index terms. The use of charge coupled devices or magnetic bubble memory devices provide good access speed. The primary function of the SM is to retrieve and update structural information of the database. This information likely to be large ($10 \times 7 - 10 \times 9$ bytes).

Index Translation Unit (IXU)

The index terms stored in the structure memory (SM) and manipulated by the structure memory information processor (SMIP) are actually represented in an intermediate form. The purpose of the IXU is to translate them into an usable form for the mass memory (MM). The other function of the IXU is the assignment and release of cluster identifiers and security atom names, on demand.

Keyword Transformation Unit (KXU)

KXU allows the structure memory first to readily identify the modules which contain the index terms of the keywords, and then to process index terms and keywords rapidly since KXU transform all information to be stored in the structure memory into fixed length fields (KXU converts the keywords into their internal representations). Keywords are sent to the KXU by the DBC's Command and Control

Processor (DBCCP). The output of the KXU is sent to the SM which retrieves index terms for the transformed keyword predicates and sends them to the SMIP. The SMIP output is interpreted by the IXU and sent to the DBCCP. This pipeline of processors results in maximum utilization of the hardware.

Structure Loop

The structure loop identifies the disk cylinders containing the actual data records. The four components of the structure loop are designed to operate concurrently.

Data Loop

The data loop is used for storing and accessing the database, for post-processing of retrieved records, and for enforcing field-level security.

The DBC design allows for a number of functionally specialized modules which can all work concurrently. A large degree of parallelism is provided within each module (including the mass memory) by employing a set of processors to simultaneously perform a content-search operation and by providing track-in-parallel read-out.

3.4.4. DIRECT

DIRECT [12] is another multiprocessor system, and it is

designed for supporting relational database management systems. This system has a multiple-instruction multiple-data stream (MIMD) architecture. It can simultaneously support both intra-query and inter-query concurrency. Microprocessors are dynamically assigned to a query depending on their priority, the type and number of relational algebra operations they contain, and the size of the relations referenced. Since DIRECT is a virtual memory machine, the maximum relation size is not limited to that of the associative memory as in some other database machines.

System architecture

The DIRECT system consists of six main components, as seen in Figure 22. These components are: a host processor, the back-end controller, a set of query processors, a set of charge coupled device (CCD) memory modules, an interconnection matrix, and one or more mass storage devices.

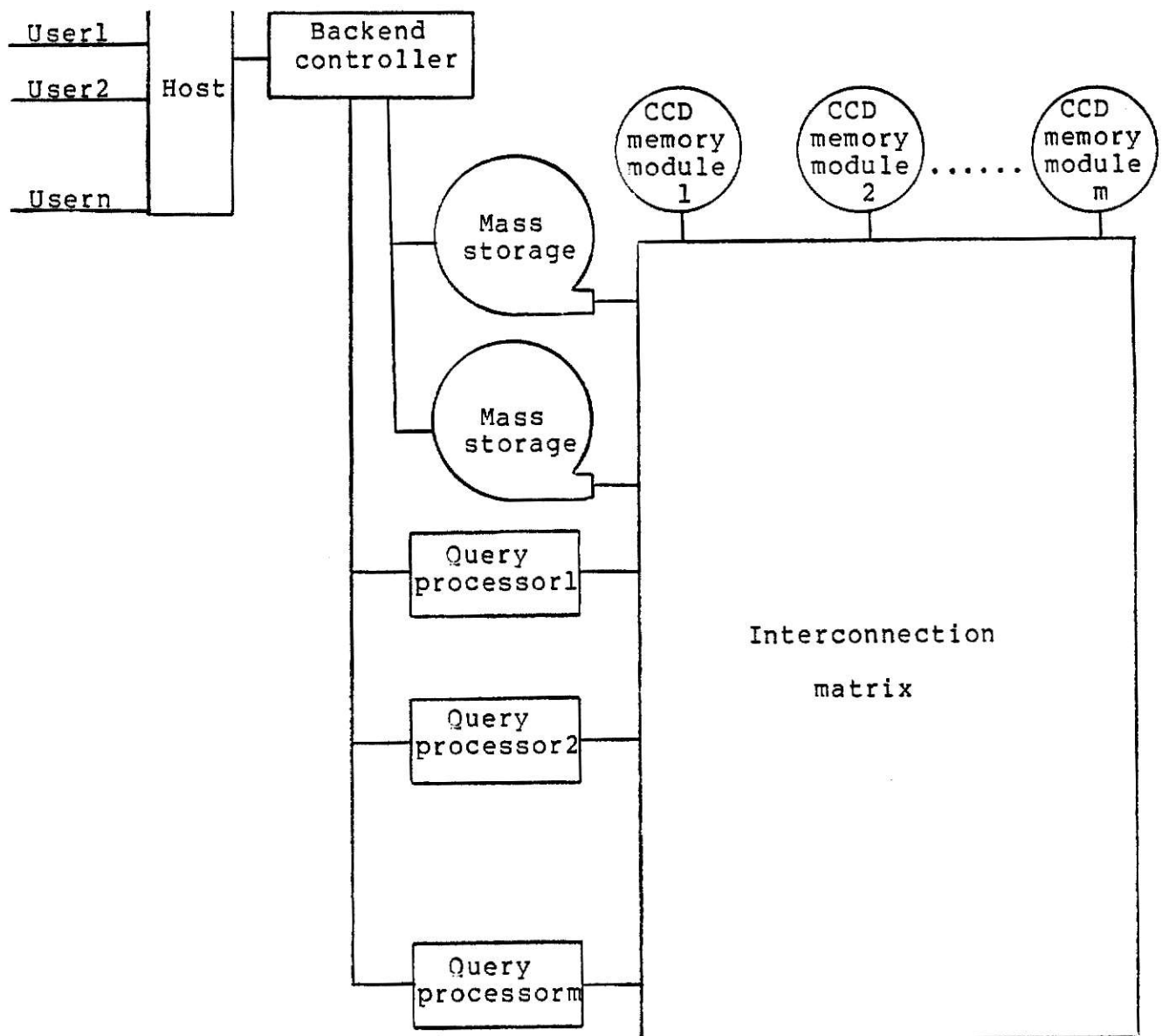


Figure 22. DIRECT System Architecture

Host processor

The host processor is a PDP 11/45 running the UNIX operating system. It handles all communications with the users. When executing a query, INGRES (a relational database system) will first compile the user query into a sequence of relational algebra operations called a "query packet" and then sends it to the back-end controller.

Back-end controller

The back-end controller is a microprogrammable PDP 11/40 which is responsible for interacting with the host processor and controlling the query processors. Once the back-end controller receives the query packet from the host, it will determine the number of query processors that should be assigned to execute the packet. If the relations which are referenced by the query packet are not currently in associative memory, the back-end controller will page portions of them in before distributing the query packet to each query processor selected for its execution. The back-end controller also handles the conflicts which can arise when two or more query processors attempt to write onto the same page simultaneously.

Query processors

Each query processor is a PDP 11/03 with 28K words memory. The function of each query processor is to execute query packets assigned by the back-end controller and

transmitted from the controller over a parallel word interface to the query processor. To facilitate the support of intra-query concurrency, relations are divided into fixed size pages. Each query processor, assigned by the controller to execute a query packet, will associatively search a subset of each relation referenced in the packet. When it finishes examining one page of a relation, it will make a request to the back-end controller for the address of the next page it should examine. To facilitate support of inter-query concurrency, the associative memory and interconnection matrix must permit two query processors, each executing different queries, to search the same page of a common relation simultaneously.

Memory modules

Memory modules are associative memories which each of them is divided into fixed size page of 16K bytes. Each page frame is constructed from eight CCD chips. The page sizes are chosen to be small for several reasons. First, more page frames will have a higher potential for concurrency. If the page size was too large, then each relation might fit on just one page and this would limit the potential concurrency to just inter-query concurrency instead of a mix of intra-and inter-query concurrency. Another reason is to minimize the amount of internal fragmentation which occurs when a relation does not fill all of the pages it occupies.

Interconnection matrix

The interconnection matrix supports the inter-and intra-query concurrency. To do so, it permits:

1. A query processor to rapidly switch between page frames containing pages of the same or different relations;
2. Two or more query processors to simultaneously search the same page of a relation;
3. All query processors to simultaneously access the same page frame.

In summary the features of DIRECT are as follows:

1. Simultaneous execution of relational queries from different users in addition to parallel processing of a single query.
2. Dynamic determination of the number of processors assigned to a query based on the priority of the query, the size of the relations it references, and the type and number of relational algebra operations included in the query.
3. Relation size is not limited by the size of the associative memory.
4. Control of concurrent updates through the use of locks on relations.

CHAPTER 4

THE FUTURE OF DATABASE MACHINES

The future of database machines is highly dependent on the new hardware technologies being introduced. We can look at the new hardware technologies in the areas of memory and secondary storage devices, processors and the entire system configuration. These areas, however, are related to each other. For example, the design of the entire system can be influenced by the available processor technology, and the design of the processors could be influenced by the available memory technology. The freedom of the system designer can also be increased by the availability of advanced components. Therefore the database computer designer must be familiar with the new hardware technologies.

In the area of storage and main memory the trend is towards hardware devices which can speed up the computerized processes (information retrieval, processing, updating, etc.), such as laser memories, semiconductor memories, bubble memories, electron beam memories, and charge coupled devices. Moving head disks will continue to be a choice as a secondary storage because of their large storage capacity and relatively low price. The bubble memories, semiconducto

memories, electron-beam memories and CCDs could also be used as a secondary storage. The fixed head disks will be used as little as possible and more likely they will be replaced by other storage devices.

In the area of processors it is quite feasible that they will be dedicated to specific database functions and their processing and speed capabilities will increase while their cost decreases. These systems will be more hardware oriented. By the use of the up-to-date semiconductor technologies it is also feasible to support a wide variety of database software - implemented - by hardware devices (often called firmware). The concurrent use of these of these processors is also important in the future database machines.

Certain important problems such as recovery from failure, concurrency control, integrity and security constraints are currently being implemented by software. The future research is anticipated in designing special processors with hardware capabilities for implementing these functions.

As an overall design the integrated system with independent functionally specialized components and concurrent capabilities of these components are feasible. The system can be very flexible in a way that it could be used as a back-end machine with a few functionally specialized processes and by adding more processors. The capability of the system would be increased. The system,

however, should be transparent to the user and be able to accept current application programs and data files while increasing performance and capability.

LIST OF REFERENCES

1. Babb, E., "Implementing a Relational Database by Means of Specialized Hardware", ACM Transactions of Database Systems, Vol. 4, No. 1, March 1979, pp. 1-29.
2. Bannerjee, Jayanta and David K. Hsiao and Richard I. Baum, "Concepts and Capabilities of a Database Computer", ACM Transactions on Database Systems, Vol. 3, No. 4, December 1978, pp. 347-384.
3. Bannerjee, Jayanta and David K. Hsiao and Krishnamurthi Kannan, "DBC - A Database Computer for Very Large Databases", IEEE Transaction on Computers, Vol. C-28, No. 6, June 1979, pp. 414-429.
4. Batchner, K.E., "STARN Series E." Proceedings 1977 International Conference on Parallel Processing, Aug 1977, pp. 140-143.
5. Baum, R.I. and D.K. Hsiao, "Database Computers - A Step Towards Data Utilities", IEEE Transactions On Computers, Vol. C-25, No. 12, December 1976, pp. 1254-1259.
6. Berra, Bruce P., "Some Problems in Associative Processor Applications to Database Management", Proceedings 1974 AFIPS National Computer Conference, Vol. 43, 1974, pp. 1-5.
7. Berra, Bruce P. and Ellen Oliver, "The Role of Associative Array Processors in Database Machine Architecture", Computer, March 1979, pp. 53-61.
8. Bray, Olin and Kenneth J. Thurber, "What's happening with database processors?", Datamation, Jan 1979, pp. 146-156.
9. Canaday, R.D. Harison, E.L. Ivie, J.L. Ryder, and L.A. wehr, "A Backend compu for Data Base Management", Communications of the ACM, Vol. 17, No. 10, October 1974, pp. 575-582.

10. Champine, G.A., "Four approaches to a Data Base Computer", Datamation, December 1978, pp. 101-106.
11. Davis, E.W., "STARAN parallel processor system software", AFIPS Conference Proceedings, Vol. 43, May 6-10, 1974, pp. 17-22.
12. Dewitt, David J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems", IEEE Transactions on Computer, Vol. C-28, No. 6, June 1979, pp. 395-406.
13. Epstein, Robert, and P. Hawthorn, "AID in the '80s", Datamation, February 1980, pp. 154-158.
14. Epstein, Robert, and P. Hawthorn, "Design decisions for the intelligent databases machine", AFIPS Conference Proceedings, Vol. 49, May 1980, pp. 237-241.
15. Heart, F.E., R.E. Kahn, S.M. Ornestein, W.R. Crowther, and D.C. Walden, "The Interface Message Processor for the ARPA Computer Network," Proceedings AFIPS Spring Joint Computer Conference, 1970, pp. 551-567.
16. Kerr, Douglass S., "Database Machines With Large Content Addressable Blocks and Structural Information Processors", Computer, March 1979, pp. 64-79.
17. Marill, T., and Stern, D., "The data computer - a network data utility", AFIPS Conference Proceedings, Vol. 44, May 1975, pp. 389-395.
18. Maryanski, F. J., "Backend Database Systems", Computing Surveys, Vol. 12, No. 1, March 1980.
19. Ozkarahan, E.A. and S.A. Schuster and K.C. Smith, "RAP - An Associative Processor for Database Management", National Computer Conference, 1975, pp. 379-387.
20. Rudolph, J.A., "A Production Implementation of an Associative Array Processo - STARAN", AFIPS Conference Proceedings, Vol. 41, 1972, pp. 229-241.

21. Schuster, S.A. and H.B. Nguyen and E.A. Ozarahan and K.C. Smith, "RAP.2 - An Associative Processor for Databases and Its Applications", IEEE Transactions on Computers, Vol. C-28, No. 6, June 1979, pp. 446-458.
22. Smith, Diane C.P. and John M. Smith, "Relational Database Machines", Computer, March 1979, pp. 28-38.
23. Su, S.Y.W. and G.J. Lipouski, "CASSM: A Cellular System for Very Large Data Bases", Proceedings International Conference on Very Large Data Bases, September 1975, pp. 456-472.
24. Su, S.Y.W., "Celluar Logic Devices: Concepts and Applications", Computer, March 1979, pp. 11-25.
25. Su, S.Y.W. and H. Chang and P. Fisher and E. Lowenthal and S. Schuster, "Database machines and some issues on DBMS standards", AFIPS Conference Proceedings, 1980, pp. 191-207.
26. Su, S.Y.W. and Le Huu Nguyen and Ahmed Emam and G.J. Lipovski, "The Architectural Features and Implementation Techniques of the Multicell CASSM", IEEE Transactions on Computers, Vol. C-28, No. 6, June 1979, pp. 430-445.
27. Thurber, K.J. and L.D. Wald, "Associative and Parallel Processors", Computing Surveys, Vol. 7, No. 4, 1975, pp. 215-255.
28. Yau, S.S. and H.S. Fung, "Associative Processor Architecture - A survey", ACM Computing Surveys, Vol.9, No. 1, March 1977, pp. 3-28

ADDITIONAL READINGS

1. Chang, H., "On bubble memories and relational data base", 4th International Conference Proceedings on Very Large Data Bases, September 13-15, 1978, pp. 207-229.
2. Chyuan S. L. and D. C. P. Smith, and J. M. Smith, "The design of a rotating associative memory for relational database applications", ACM Transaction On Database Systems, Vol. 1, No. 1, March 1976, pp 53-65.
3. Coulouris, G.F., "Towards Content-addressing in Data Bases", Computer Journal, Vol. 15, February 1972, pp. 95-98.
4. DeFiore, C.R. and N.J. Stillman and P.B. Berra, "Associative Techniques in the Solution of Data Management Problems", Proceedings 1971 ACM National Conference.
5. DeFiore, L.R., P.B. Berra, "A Data Management System Utilizing an Associative Memory", AFIPS Conference Proceedings, Vol. 41, 1972.
6. Holland, Robert H. and Ann Mich, "Improve Information Access With the Database Machine", Data Communications, March 1980, pp. 95-101.
7. Hsiao, David K., "Database Machines Are Coming, Database Machines Are Coming!", Computer, March 1979, pp. 7-9.
8. Langon, Glen G., "Database Machines: An Introduction", IEEE Transactions on Computers, Vol. C-28, No. 6, June 1979, pp. 381-382.
9. Love, H.H, "An Efficient Associative Processor Using Bulk Storage", Proceedings 1973 Sagamore Computer Conference on Parallel Processing, 1973, pp. 103-112.
10. Maryanski, F.J., "A survey of developments in distributed data base management systems", Computer,

February 1978, pp. 28-37.

11. Maryanski, F.J., V.E. Wallentine, "A simulation Model of a Back_end Data Base Management System", Technical Report, Computer Science Dept., Kansas State University, Manhattan, Kansas, April 1976.
12. Minsky, N., "Rotating Storage Devices as Partially Associative Memories", AFIPS Conference Proceedings, Vol. 41, 1972.
13. Moulder, R., "An Implementation of a Data Management System On an Associative Processor", AFIPS Conference Proceedings, Vol. 42, June 1973, pp. 171-176.
14. Parhami, B., "A Highly Parallel Computing System for Information Retrieval", AFIPS Conference Proceedings, Vol. 41, December 1972, pp. 681-690.
15. Roberts, L.G. and B.D. Wessler, "Computer Network Development to Achieve Resource Sharing," Proceedings AFIPS Spring Joint Computer Conference, 1970, pp. 543-549.

CONCEPTS AND CAPABILITIES OF DATABASE MACHINES

by

NASSRIN TAVAKOLI

B.S., Southeast Missouri State University, 1978

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements of the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1981

ABSTRACT

A number of major problems have been faced by the database designers for a long time. These problems are due to the limitation and the nature of conventional computers, and can only be solved by introducing new architectural concepts. Database machine definitely seems to have a place in solving these problems. The field is developing because of the increased throughput in processing database requests at a competitive cost and the parallelism and content addressing capabilities.

This report begins with the description of the limitations of conventional computers for database applications.

A larger portion of this report deals with the concepts and capabilities of database machines in a categorized manner. The categories are based on the architecture, objectives and characteristics of the different database machines.

In the last part some ideas are given about the future trends.