

ARCHITECTURAL FEATURES OF WCRC -  
A DATA BASE COMPUTER



By  
Surya R. Dumpala, B.Sc., B.E.

A Thesis  
Submitted to the School of Graduate Studies  
in Partial Fulfilment of the Requirements  
for the Degree  
Master of Engineering

McMaster University  
Hamilton, Ontario  
March, 1981

MASTER OF ENGINEERING  
(Electrical Engineering)

McMASTER UNIVERSITY  
Hamilton, Ontario

TITLE: Architectural Features of WCRC - A Data Base  
Computer

AUTHOR: Surya R. Dumpala  
B.Sc. (Andhra University)  
B.E. (Indian Institute of Science)

SUPERVISOR: Dr. Sudhir K. Arora

NUMBER OF PAGES: xiv, 259

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my supervisor, Dr. Sudhir K. Arora, for his constant guidance and helpful discussions throughout the course of this work. I also extend my thanks to Dr. D. P. Taylor and Dr. J. B. Anderson for serving as the supervisory committee.

I would like to express my appreciation to McMaster University for financial support.

Finally, the prompt and excellent typing services provided by Ms. Linda Hunter are gratefully acknowledged.

## ABSTRACT

Several data base machine architectures have been proposed in the past few years. The next generation of these machines must support different data models on the same physical data simultaneously as envisaged in the ANSI/X3/SPARC report or the coexistence model.

This thesis presents the architectural features of one such data base machine called Well Connected Relational Computer (WCRC). The overall architecture and the facilities to the user as well as the DBA have been described. A framework for the conceptual level and detailed design of the internal level are reported. The algorithms for schema conversion and view translation have been developed. The conceptual level language, WCRL is extended to accommodate data definition, data manipulation and storage definition facilities. A high level language, DBAL, is developed for the DBA. Two binary storage structures (Pseudo Canonical Partitions (PCP's - Options I and II) have been reported for storing the data at the internal level. They radically differ from the conventional n-ary relational storage structure. A machine oriented language (WCRML) is developed to directly execute the data base instructions in hardware on the PCP storage structures. The basic hardware organization of the internal level along with special hardware units for some data base functions such as join and sort have been reported. Finally, a performance

evaluation of WCRC storage structures has been presented. The results indicate that WCRC requires lesser storage and offers faster query response time when compared to architectures based on n-ary relation storage.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
TABLE OF CONTENTS	vi
LIST OF FIGURES	x
LIST OF GRAPHS	xiv
CHAPTER I - INTRODUCTION	1
CHAPTER II - BACKGROUND AND PROBLEM DEFINITION	4
2.1 Introduction	4
2.2 The Data Models	4
2.2.1 The Relational Model	5
2.2.2 The Network Model	9
2.2.3 The Hierarchical Model	9
2.3 Associative Processors and Logic-Per-Track Devices	11
2.4 Data Base Systems Implemented on Special Purpose Hardware	17
2.5 Problem Definition	20
CHAPTER III - OVERALL ARCHITECTURE OF WCRC	22
3.1 Basic Organization	22
3.2 Query Processing	26

	<u>Page</u>
3.3 Data Base Facilities in WCRC	29
3.3.1 The External Languages	29
3.3.2 Security and Integrity	30
3.3.3 View Generation and Updates Through Views	34
3.3.4 Concurrency	40
3.3.5 Data Base Growth	40
 CHAPTER IV - THE CONCEPTUAL LEVEL	 42
4.1 Introduction	42
4.2 Choice of the Data Model	43
4.2.1 Entity-Relationship Model	44
4.2.2 Derivation of Other Data Models	46
4.3 Choice of the Data Language	52
4.3.1 WCRL - Data Retrieval Language	54
4.3.2 Data Definition Language (DDL) for WCRL	60
4.3.3 Data Manipulation Language (DML) for WCRL	65
4.3.4 Storage Definition Language (SDL) for WCRL	68
4.4 Data Base Administrator Language (DBAL)	71
4.4.1 DBAL - Data Definition (DDL)	72
4.4.2 DBAL - Data Manipulation (DML)	77
4.4.3 DBAL - Storage Definition (SDL)	83
4.5 Query Translation	84
4.6 Query Analysis	92
4.7 View Translation, Consistency and Update Strategy	99

	<u>Page</u>
4.7.1 Mapping Relational Views into E-R Model	100
4.7.2 Mapping Network Views into E-R Model	107
4.7.3 Mapping Hierarchical Views into E-R Model	110
4.7.4 View Consistency and Updates	117
4.8 Summary	129
 CHAPTER V - THE INTERNAL LEVEL	 130
5.1 Introduction	130
5.2 The Data Structures	131
5.2.1 PCP - Option I	131
5.2.2 PCP - Option II	135
5.3 A Comparison of Options I and II	138
5.4 WCRML Instruction Set	144
5.4.1 Instructions Executed by the Cell Processor	144
5.4.2 Instructions Executed by a Query Processor	151
5.4.3 Translation of WCRL into WCRML Instructions	154
5.5 Cell Hardware	159
5.5.1 Circulating Memory (CU)	159
5.5.2 Track Format Identification Unit (TFIU)	163
5.5.3 Buffer Unit (BU)	167
5.5.4 Search Unit (SU)	169
5.5.5 Arithmetic and Logic Unit (ALU)	171
5.5.6 RAM Logic Unit (RLU)	174
5.5.7 I/O Unit (IOU)	174
5.5.8 Control Unit (CU)	176



	<u>Page</u>
5.6 Cell Executed Instruction Times	180
5.7 Query Processor Hardware	184
5.7.1 Boolean Evaluator	184
5.7.2 Join and Sort Unit	188
5.7.3 RAM POOL	196
5.7.4 Controller	198
5.8 Summary	198
 CHAPTER VI - PERFORMANCE EVALUATION	 199
6.1 Introduction	199
6.2 Storage Requirements	199
6.3 Retrieval Time	217
6.4 Comparison of Overall Retrieval Times of GDBMS and WCRC	224
6.5 Update Time	230
6.6 Overall Comparison of GDBMS and WCRC	230
6.7 Summary	232
 CHAPTER VII - DISCUSSION AND FUTURE RESEARCH	 233
7.1 Discussion	233
7.2 Future Research	235
 REFERENCES	 237
 APPENDIX A	 243
 APPENDIX B	 250
 APPENDIX C	 256

LIST OF FIGURES

<u>Figure</u>	<u>Title</u>	<u>Page</u>
2.1	The Relation - Graduate Students	6
2.2	A Network of Record Types and Links	10
2.3	A Hierarchical Data Base	12
2.4	An Associative Memory	13
2.5	Logic Per Track Devices	15
3.1	Overall Architecture of WCRC	23
3.2	The Stages of Query Processing	28
3.3	The User Access Table (UAT)	32
3.4	Some Views in Relational Model	36
3.5	Some Views in Hierarchical Model	37
3.6	Some Views in Network Model	38
4.1	E-R Diagram of Corporate Data Base	47
4.2	Relational Equivalent of the Corporate Data Base	49
4.3	Network Equivalent of the Corporate Data Base	51
4.4	Hierarchical Equivalent of the Corporate Data Base	53
4.5	Hierarchical Tree Corresponding to a N:M Relationship	53
4.6	Illustration of a WCR, an EWCR and a CP	56
4.7	Conceptual Schema of an Example Data Base	75

<u>Figure</u>	<u>Title</u>	<u>Page</u>
4.8 a	An Example Data Base	79
4.8 b	The Data Base After Adding Entity 'City'	79
4.8 c	The Data Base After Splitting 'Phone'	80
4.8 d	The Data Base After Shift Operations	80
4.9	E-R Diagram of a Corporate Data Base	85
4.10	The External Schema in Relational Model	86
4.11	The External Schema in Network Model	86
4.12	The External Schema in Hierarchical Model	87
4.13	The PCP's Corresponding to Fig. 4.9	87
4.14	Permutations of Type II b Condition	95
4.15	Equivalent Forms of Type II b Conditions	96
4.16	A Relational View of a Hypothetical Data Base	106
4.17	The E-R View Corresponding to Fig. 4.16	108
4.18	A Network View of a Hypothetical Data Base	111
4.19	The E-R Model of the Network View in Fig. 4.18	112
4.20	A Hierarchical View of a University Data Base	115
4.21	a) A Graph Corresponding to Fig. 4.20	116
	b) The E-R Model Corresponding to Fig. 4.20	116
4.22	Behaviour of Binary Relationship Relations Under Natural Join	121
5.1	Track Format of PCP - Option I	133
5.2	PCP's on a Track - Option I	134
5.3	Track Format of PCP - Option II	137

<u>Figure</u>	<u>Title</u>	<u>Page</u>
5.4	PCP's on a Track - Option II	139
5.5	An Example Showing PCP's Corresponding to an E-R Schema	140
5.6	(a) An Example Data Base (b) The PCP's	155
5.7	Overall Organization of a Cell	160
5.8	Storage Format	162
5.9	Gap Indicators	164
5.10	TEM, TBM and DS Indicators	166
5.11	Bit Sequence Indicators and Flag Indicators	168
5.12	Buffer Unit	170
5.13	a) Overall Search Unit;	172
	b) One Comparator Unit	172
5.14	ALU Unit	173
5.15	RAM Logic Unit	175
5.16	I/O Unit	175
5.17	Control Unit	177
5.18	$\mu$ -Programmed	179
5.19	Organization of a QP	185
5.20	Boolean Evaluator	187
5.21	Join and Sort Unit	190
5.22	Associative Memory (Data Part)	192
5.23	Flow Chart to Find Maximum and Minimum	194
5.24	Cartesian Product	197
6.1	A Hierarchy of Relations With n-Level Embedding	201

<u>Figure</u>	<u>Title</u>	<u>Page</u>
6.2	ANF in Associative Memory	202
6.3	An Example Showing How ANF is Stored as PCP's	205
6.4	PCP RAM POOLS at Level $i$ and $i+1$	221

LIST OF GRAPHS

<u>Graph</u>	<u>Title</u>	<u>Page</u>
6.1	Storage vs. $l$	
6.2	Storage vs. $N_D$	
6.3	Storage vs. $n$	
6.4	Storage vs. $s$	
6.5	Storage vs. $a$	
6.6	Number of Interrogations vs. Level of Embedding	

## CHAPTER I

### INTRODUCTION

In recent years, there has been a rapid growth in the usage of Data Base Management Systems (DBMS's). They are being widely used in several areas such as industry, business, government, library management, artificial intelligence and natural language manipulation. These areas primarily involve manipulation of information, rather than complicated numerical computations.

The early DBMS's were implemented on conventional Von Neumann type machines which are primarily intended for numerical tasks. These systems suffer from the following drawbacks:

1. The non-numeric capabilities such as search, retrieve and update, desirable by a DBMS have been provided on the conventional machines at prohibitively high software cost and complexity.
2. With the increasing size of the data bases, increasing complexity of data structures and user's demands becoming more sophisticated, the software overhead to support these systems has further grown in complexity and cost.
3. It is difficult to achieve a high degree of data independence and provide a framework to support multiple user views of the data base using conventional architectures. Data independence is a measure of how well applications are insulated from changes to the physical

organization of data - "Physical Data Independence". Also, different users must be able to view the data independent of each other - "Logical Data Independence".

The increasing software overhead of these systems calls for a machine architecture which can support some of the data base functions at the hardware level. The rapid decrease in the cost of hardware with technology makes it more attractive for developing such new hardware-oriented architecture. In addition, achieving logical and physical data independence calls for organizing the database at various levels. The idea of levels of a DBMS has been formally proposed as a framework for DBMS architecture in the ANSI/X3/SPARC Report [ANSI 75]. This report proposes three levels for a DBMS namely, the external, the conceptual and the internal. The external level corresponds to different user views and applications which may be based on different data models. The conceptual level represents the data model which corresponds to a 'stable' and common view of the data. The internal level deals with the organization of physical storage of the data. Implementing such a multilevel architecture on a conventional machine by software will suffer from the drawbacks discussed earlier. Therefore, it is worthwhile to develop a computer architecture to meet these needs.

This report addresses the problem of designing a data base computer architecture (WCRC) that can support multiple user environment within the ANSI/SPARC framework utilizing specialized hardware. To the best of our knowledge, there is only one other project in the world addressing this problem [DOGAC 80]. The relevant background and scope of the problem are



discussed in Chapter II. An overview of the proposed data base machine is presented in Chapter III. Chapter IV and V discuss salient features of the three levels of the computer. In addition, performance analysis of the proposed system in comparison with Generalized Data Base Management System (GDBMS) [DOGAC 80] and some previous work [DEFIORE 74] is presented in Chapter VI. Finally, a discussion of this research, together with suggestions for future research, is presented in Chapter VII.

## CHAPTER II

### BACKGROUND AND PROBLEM DEFINITION

#### 2.1 Introduction

The problem addressed in this report is the design of a data base computer with special hardware for supporting different data models. Therefore, the literature review will focus on non-numeric computer architectures, specialized hardware systems that support some DBMS functions and schemes that support multiple user views. However, before proceeding with the literature survey, two subsections are presented to acquaint the reader with the concepts and terminology of the three widely-used data models and associative processors, which are essential for further discussion.

#### 2.2 The Three Data Models

A data model is an abstraction of the "real world". It forms the basis for representing and manipulating the reality as perceived by a designer. Within a model of the world, similar things are usually grouped into classes of objects called object types [SCHMID 75]. An example of an object type is employees. An object type is described by listing its characteristics. For instance, the object type employees has characteristics such as employee number, name, age and salary. Object types that have an independent existence and can be meaningfully consid-

ered by themselves are interpreted as sets of entities or entity sets. The characteristics of an entity set are known as attributes. The correspondence or mapping between attributes of an entity set is called an attribute relationship. The relationship (or mapping) between entity sets is referred to as an association. For example, an employee's name and salary have an attribute relationship between them. On the other hand, the two entity sets such as employees and department, which exist independently, have an association "work in" between them.

A number of data models have been suggested in the literature to capture the entity sets and relationships existing in the real world. The three most important such models are the relational, the network, and the hierarchical. The distinctive characteristics of these models are briefly discussed in the following sections.

### 2.2.1 The Relational Model

The relational model is based on the mathematical theory of relations [CODD 70].

Definition 2.1: Given sets  $D_1, D_2, \dots, D_n$  (not necessarily distinct),  $R$  is a relation if it is a subset of the Cartesian Product  $D_1 \times D_2 \times \dots \times D_n$ . That is,  $R$  is a set of ordered tuples  $\langle d_1, d_2, \dots, d_n \rangle$  such that  $d_1$  belongs to  $D_1$ ,  $d_2$  belongs to  $D_2$ , ...,  $d_n$  belongs to  $D_n$ . Sets  $D_1, D_2, \dots, D_n$  are called the domains of  $R$  and the value  $n$  is called the degree of  $R$ .

Alternatively, a relation may be viewed as a two-dimensional table as shown in Fig. 2.1. Each row is referred to as one tuple and

STUDENT #	NAME	DEPARTMENT	PROGRAM
7929420	Johnson	Electrical	M. Eng.
7984185	Smith	Mechanical	Ph.D.
7842535	Parker	Civil	M. Eng.
7926476	Miller	Electrical	M. Eng.
7891887	Brown	Electrical	Ph.D.

Fig. 2.1 The Relation - Graduate Students

each column is called an attribute. The domain is the set of values that can appear in that attribute. Further, the table has the following properties: 1) No two rows are identical, 2) The ordering of rows and columns is unimportant.

Definition 2.2: An attribute whose domain contains values which uniquely identify the tuples of the relation is called the primary key. Such a key is said to be nonredundant if it is either a simple domain or a combination such that none of the simple domains in the combination is superfluous in uniquely identifying the tuple.

Definition 2.3: In a Relation R, the domain Y is functionally dependent on domain X if and only if (iff) each X-value is associated with exactly one Y-value at any particular instant. Domain Y is fully functionally dependent on domain X if it is functionally dependent on X and not functionally dependent on any subset of X.

In Fig. 2.1, STUDENT # is the key and domains DEPARTMENT and PROGRAM are functionally dependent on NAME.

Definition 2.4: A relation is said to be in first normal form (1NF) if each of its domains contains only atomic values. A relation is said to be in second normal form (2NF) iff it is in 1NF and the non-key domains of R, if any, are fully functionally dependent on the primary key of R. A relation is in third normal form (3NF) iff it is in 2NF and the non-key domains of R, if any, are 1) mutually independent (i.e. not functionally dependent) and, 2) fully functionally dependent on the primary key of R.

A further discussion on dependencies and normalization may be found in [TSICHRITZIS 77] and [ULLMAN 80].

In the relational data model, both the entities in the real world data and the associations among them are represented by relations. New relations are generated from the existing relations in the data base, by using relational operations. There are five basic operations that serve to define relational algebra:

1. Projection: It is the operation whereby certain specified columns of a relation R are selected and the duplicate rows are removed from the resultant relation.
2. Selection: This operation selects a set of tuples in R that satisfy a given clause F, where F involves operands that are constants, the arithmetic comparison operators (>, =, < etc.) and the logical operators (AND, OR and NOT). For example, in Fig. 2.1, F may be NAME = 'Smith' AND PROGRAM = 'Ph.D.'.
3. Join: The Join of two relations R and S over an attribute(s) is the set of tuples formed by taking cross product of those tuples from R and S satisfying the join relationship/condition. It is used to combine two or more relations.
4. Union: The union of relations R or S is the set of tuples that are in R or S or both.
5. Difference: The difference of relations R and S is the set of tuples in R but not in S.

Thus, the relational model provides a simple data structure, the

relation, as a basis of the data base and enables the user to define flexible data relationships using operations such as projection and join.

### 2.2.2 The Network Model

The network model consists of 'record types' and 'links'. Record types are used to represent the relationships among the attributes of an entity set. Links are employed to represent the associations between entity sets. [TSICHRITZIS 77].

In general, there are no restrictions on the relationships represented by the links. They can be 1:1, 1:N or N:M mappings. Links are always labelled, as they are concrete objects representing associations in the network data model. An example of network model of a data base involving DEPARTMENTS, TEACHERS, COURSES and STUDENTS is shown in Fig. 2.2 illustrating record types and links.

### 2.2.3 The Hierarchical Model

The hierarchical data model is a special case of the network model where all the links are of the type 1:N. It is based on a hierarchical definition tree which is nothing but an ordered tree. Every node in the tree represents a record type and every arc stands for a link. There can be at most one arc between any two nodes of the tree. The relative order of the records (nodes) indicates the relationship among them. Except the root node records, no other records can have an independent occurrence (i.e. without links to

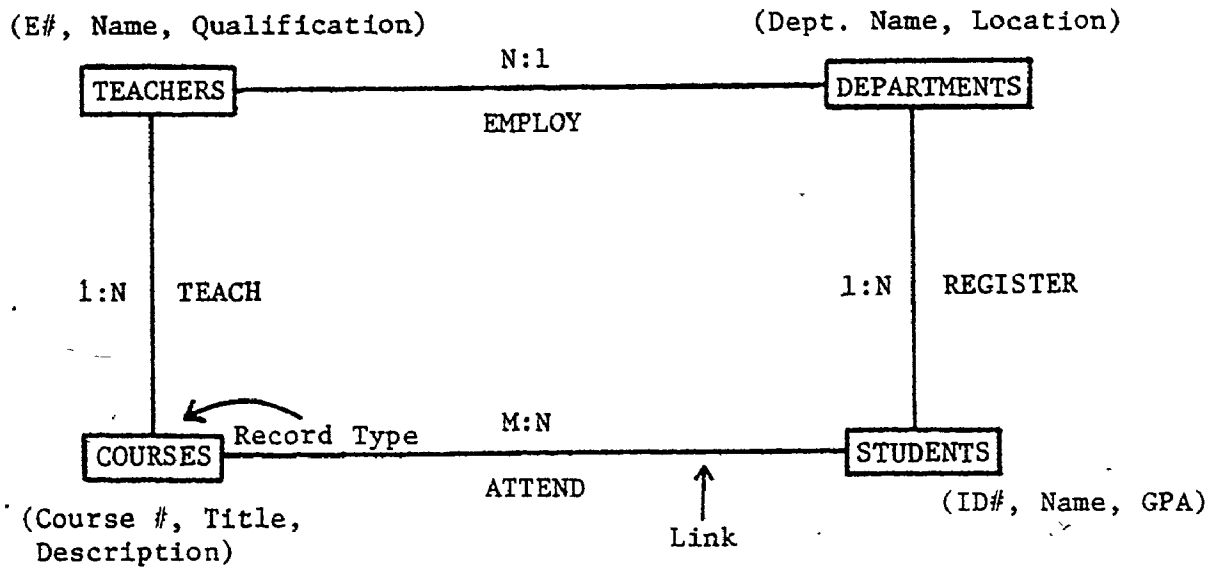


Fig. 2.2 A Network of Record Types and Links



other records). A collection of such disjoint trees constitutes a hierarchical data base. A hierarchical path in the data base is a sequence of records starting at a root record and ending at a dependent node with all the intermediate nodes exhibiting parent-child relationship alternatively.

An example of a hierarchical data base is shown in Fig. 2.3 illustrating the characteristics of the model [DATE 76].

In the next section, associative processors and logic-per-track devices are introduced to facilitate further discussion.

### 2.3 Associative Processors and Logic-Per-Track Devices

Associative processors are also known as Content Addressable Parallel Processors. In contrast to Co-ordinate Addressed Memory, the content addressable memory involves accessing of data based on the contents of the memory cells rather than the physical address of the cells. They are capable of performing parallel search operations such as exact match, maximum, minimum, less than, greater than, and the Boolean operations. Some associative memories are also capable of performing arithmetic operations. The terms associative memories and associative processors are used interchangeably in the literature [PARHAMI 73].

Figure 2.4 shows a typical associative memory. It consists of an array which contains the data; a comparand register which holds the argument that will be used against the data in the array for searching; and a mask register which enables or inhibits bit slices of the array for a given operation. The response registers are employed to record

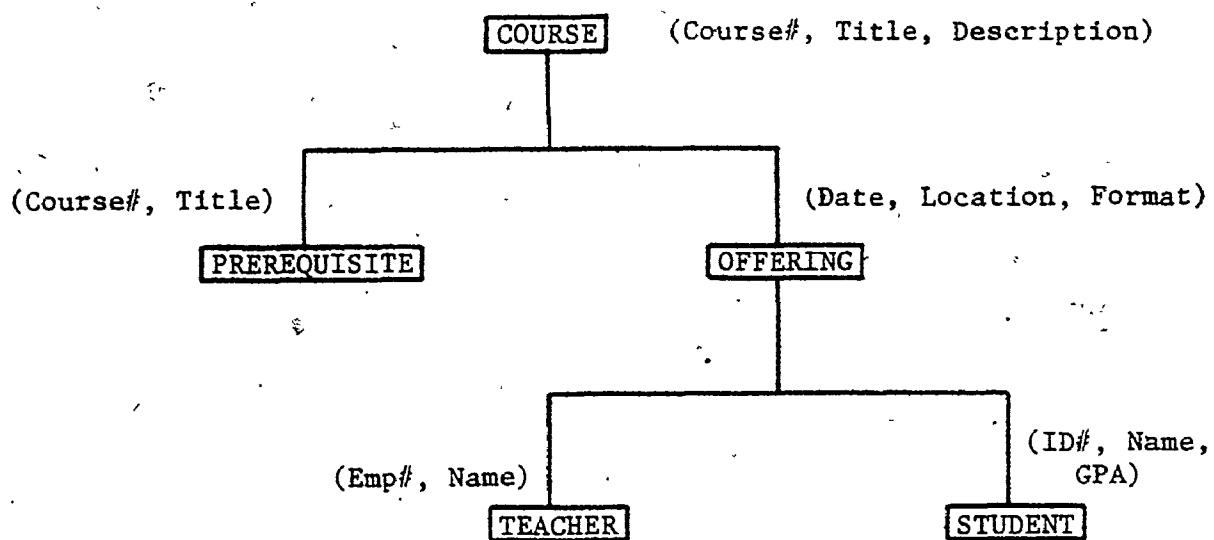


Fig. 2.3 A Hierarchical Data Base

Comparand

EE

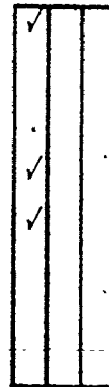
Mask

000000 11 000000

Array

Johnson	EE	M. Eng.
Smith	ME	Ph.D.
Parker	CE	M. Eng.
Miller	EE	M. Eng.
Brown	EE	Ph.D.
Blake	ME	M. Eng.
Clark	CS	Ph.D.
Jones	CS	M. Eng.

x y z



Response Registers

Fig. 2.4 An Associative Memory

the search results, perform boolean operations and provide a word selection capability. For example, consider the query where all students registered in EE have to be reported. This query is processed by placing EE in the comparand register, loading the mask register with ones in the corresponding positions, and executing an exact match search on the memory. All the words that satisfy match condition will be marked in the response register x. Any further search criteria may be similarly processed using the other response registers and performing boolean operations on the marked bits of these registers. These marked words may subsequently be retrieved from the memory array.

Further information concerning the operation of associative memories can be found by consulting references [PARHAMI 73], [PARKER 71] and [FOSTER 76] .

The content addressable memories are more expensive compared to the co-ordinate addressed memories, because of their associated logic at the cell level. The recent LSI technology is expected to bring down the cost of solid-state associative devices.

The other type of memories which have search capabilities similar to associative memories are logic-per-track devices. They differ from conventional rotating disks in their features such as logic associated with each track of the fixed head of the rotating device.

A typical logic-per-track device is shown in Fig. 2.5. It consists of four major components: Memory elements, processing elements, controller and I/O mechanism. The tracks of disks and drums, CCD shift

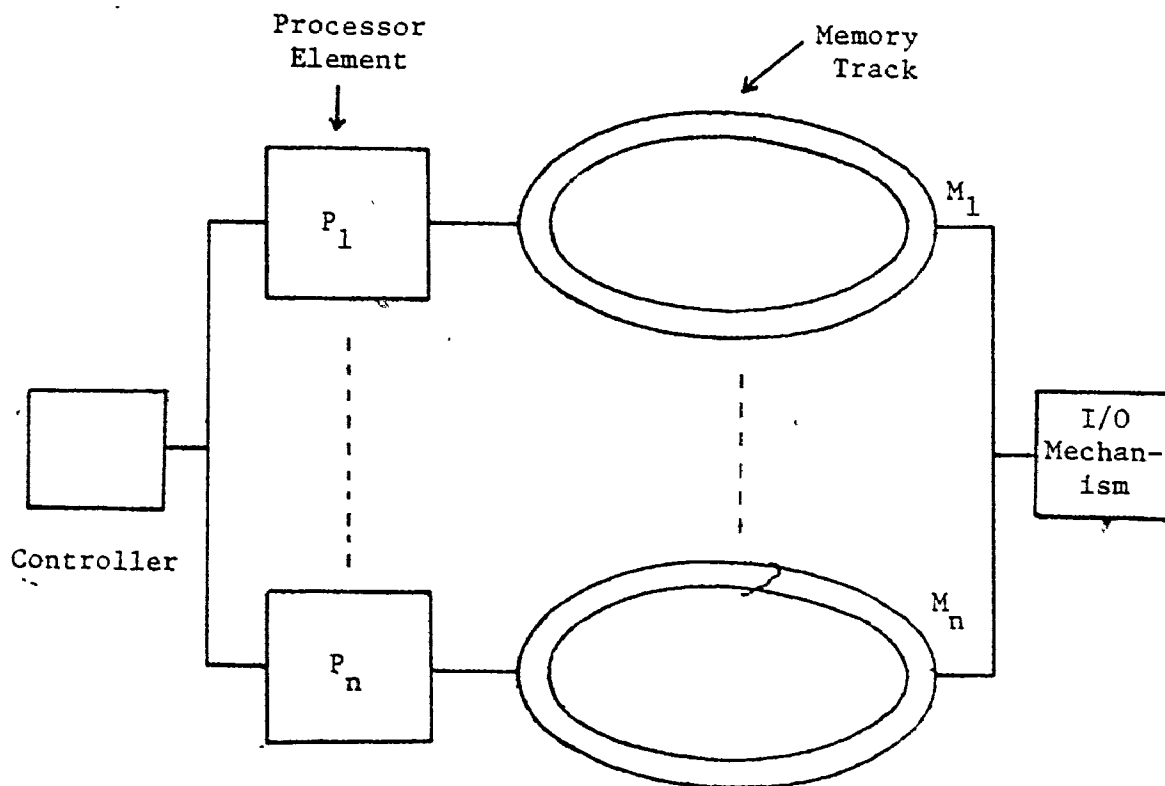


Fig. 2.5 Logic-Per-Track Devices

registers, magnetic bubble memories or any other delay line technologies may be used as the circulating memory elements. The processing elements are capable of performing search and select operations, arithmetic and logic operations, data processing operations such as 'SUM', 'AVERAGE', 'MIN', etc., and data base operations such as read, delete, modify and insert. The controller co-ordinates all the processors and controls the execution of identical instructions on several processors simultaneously. Thus, a logic-per-track device can be classified as a SIMD (single instruction multiple data stream) architectures. The I/O mechanism is responsible for moving data in and out of the memory elements.

The existing cellular logic-per-track devices vary widely, based on the facilities provided such as inter-processor communication, parallel or serial I/O and type of technology used for memory elements. These systems are also classified as partially associative memory systems since associative search capabilities and the required logic are built into the processor cells, not the data cells. These devices provide a cost effective alternative to the associative processors.

The storage organizations described in this section have contributed significantly to non-numeric processing and these techniques have influenced a number of designs for the data base computers. In the next section, some systems implemented with nonconventional computer architectures are examined.

#### 2.4 Data Base Systems Implemented on Special Purpose Hardware

A number of machine architectures have been proposed in the literature which address the problem of implementation of Data Base Management Systems on special purpose hardware. However, none of these systems provide the facilities to support different data models simultaneously.

In [LIPOVSKI 78] and [SU 79] the architecture of CASSM is presented. This is a back-end computer designed using top-down approach for efficient implementation of hierarchical model. The data is stored on memory cells as linear files. The system is implemented on head-per-track disks utilizing an array of logic cells which enables parallel processing of data. The system can handle one query at a time. Each processor cell can communicate with the neighbouring cells. Data retrieval is achieved by content addressing and location addressing.

The architecture of RAP is discussed in [OZKARAHAN 75]. A recent version of this machine called RAP-2 is described in [SCHUSTER 79]. Both are back-end processors for implementing relational data model. The design incorporates a parallel array of associative cellular processors which is driven by a Central Controller and a Statistical Processor. The data is stored on rotating logic-per-track memory devices. In the earlier version of RAP, all tracks are of the same length and the cells are allowed to communicate among themselves. But in the later version, inter-cell communication is restricted through the controller and the tracks are allowed to be of variable length. The system is of

SIMD type and it can handle only one query at a time. Data retrieval is accomplished by content addressing.

The Data Base Computer (DBC) is another back-end computer which can be used to implement network, relational or hierarchical data bases [BANNERJEE 79]. There are primarily six functional components: The Data Base Command and Control Processor (DBCCP), the Post Processor (PP), the Mass Memory (MM), the Structure Memory (SM), the Structure Memory Information Processor (SMIP) and the Index Translation Unit (IXU). The data is stored on disk-type mass memory with content addressing capability. An index to the mass memory is maintained in the CCD based SM. The DBCCP translates the data base commands into lower level commands for MM and co-ordinates all the other units. SMIP performs set operations such as AVERAGE, COUNT and SUM, while IXU is responsible for finding the address of the partition(s) of data required to answer a query. The system can handle one query at a time. Clustering and Indexing techniques are used to minimize the main memory search time.

In [DEWITT 79], the architecture of DIRECT is presented. This is a back-end computer for supporting relational data bases. The data is organized into fixed size pages on the mass memory and retrieved into rotating CCD tracks when required. This system can handle several queries simultaneously. It provides the facilities of shared reading to several query processors and single writing to one processor employing a lock on the data page. Retrieval from the CCD memory is by content addressing.

The architecture of RARES is discussed in [LIN 76]. RARES



architecture is tailored to support an "optimised relational interface such as SQUIRAL", as described by the authors. It uses search logic attached to head-per-track storage devices. It performs search and selection operations at the storage device level and provides a means for high output rate of sorted tuples. It uses the "orthogonal" storage layout which provides two advantages; reduction in storage capacity required by search logic and high output rate of selected tuples.

The Relational Associative Computer System (RELACS) is another associative system with special hardware for implementing a relational data base [OLIVER 79]. This system has five main functional units; the Dictionary/Directory Processor (DDP), the Associative Query Translator (AQT), the Mass Storage Device (MSD), the Output Buffer (OB) and the Associative Units (AU $\phi$ , AU1). It can support a large relational data base with no size restriction as it uses a specially designed associative processor to search relations. Unlike the other logic-per-track processors, this processor does not search the entire memory for answering a query. The DDP retrieves the description of the required data and passes it to the AQT along with the sequence of instructions. The AQT translates these instructions into a set of low level commands executable by the AU's, which can perform content searching.

Up to this point, the discussion has been about the systems which are designed for implementing the entire data base environment. In addition to these, there has also been research in the area of special processors for implementing specific data base functions (list mergers, search processors, etc. ...). In [LEILICH 78], the archi-

ecture of Search Processor is described. This processor is capable of performing selection and restriction on data stored on the disks. It uses 14 'search modules' to handle several queries at a time. There is no content addressing of the data on the disks. In [HOLLAR 79] a design for list merging network is discussed. It uses a number of processors combined in the form of a network to process complex boolean expressions on sorted lists without producing intermediate results.

There have also been some theoretical models proposed for supporting multilevel architectures. One such model for relational data bases is suggested in [BRACCHI 74]. The model proposes five logical levels which include the schema, the user subschema and the related logical and physical mappings. The data independence, flexibility and optimisation features of the model are highlighted.

In [KLUG 78] a scheme for supporting different data models at user level is suggested. It employs a simplified network model at the conceptual level from which the user views are derived by using mappings. A general discussion on how user queries may be translated into queries on conceptual model is presented. However, no machine architecture to support this scheme has been put forward.

## 2.5 Problem Definition

From the previous discussion, it is obvious that there is a need for a data base machine architecture which can support different data models simultaneously on the same physical data in conformity with the ANSI SPARC Proposals [ANSI 75]. To the best of our knowledge,

there is only one project in the world addressing this problem - GDBMS. [DOGAC 80]. Here the authors simulate the ANSI SPARC proposals or the co-existence model [NIJSSSEN 76]. The RAP machine is used at the internal level and the conceptual and external levels are simulated by software.

In this report, we study features of a machine architecture (WCRC) [ARORA 81] to meet the above-mentioned requirements and compare our approach to GDBMS. The machine is one hardware version of the ANSI SPARC architecture. It has an external level, a conceptual level and an internal level. It can handle several queries simultaneously. It can support three major data models, network, relational and hierarchical simultaneously on the same data at the internal level. It may be used as a back end to a host computer or as an independent data base computer to do non-numeric processing.

The overall architecture, the features of the external level and the languages used at the user level are discussed in Chapter III. The salient features of conceptual level including the language, the model, the schema translation and the Data Base Administrator (DBA) facilities are discussed in Chapter IV. The data structures, the hardware primitives and the associated hardware of the internal level are discussed in greater detail in Chapter V. The performance evaluation based on the internal level is worked out in Chapter VI. Furthermore, the performance of the system is compared with the RAP system and Defiore's Associative Scheme [DEFIORE 74].

## CHAPTER III .

### OVERALL ARCHITECTURE OF WCRC

#### 3.1 Basic Organization

WCRC, Well Connected Relational Computer is an independent data base computer intended for non-numeric processing. As shown in Fig. 3.1, the system consists of three major blocks corresponding to the three levels: **the** External Processor, the Conceptual Processor and the Internal Processor.

The External Processor performs the following main functions:

- i) Queuing of jobs (or queries)
- ii) Priority encoding of queries
- iii) Translating queries from user language into conceptual level language (WCRL)
- iv) Security checking to protect the data base from unauthorized operations.

It has three memory areas: The User Work Area (UWA), The Interface Buffer Area (IBA) and The Processor Memory Area (PMA). A portion of UWA is allotted to each user as his/her work space. In this area, the user can define his own view, a part of the data base as seen by him and specify the constraints on it such as granting other users to use this view or some additional integrity constraints.

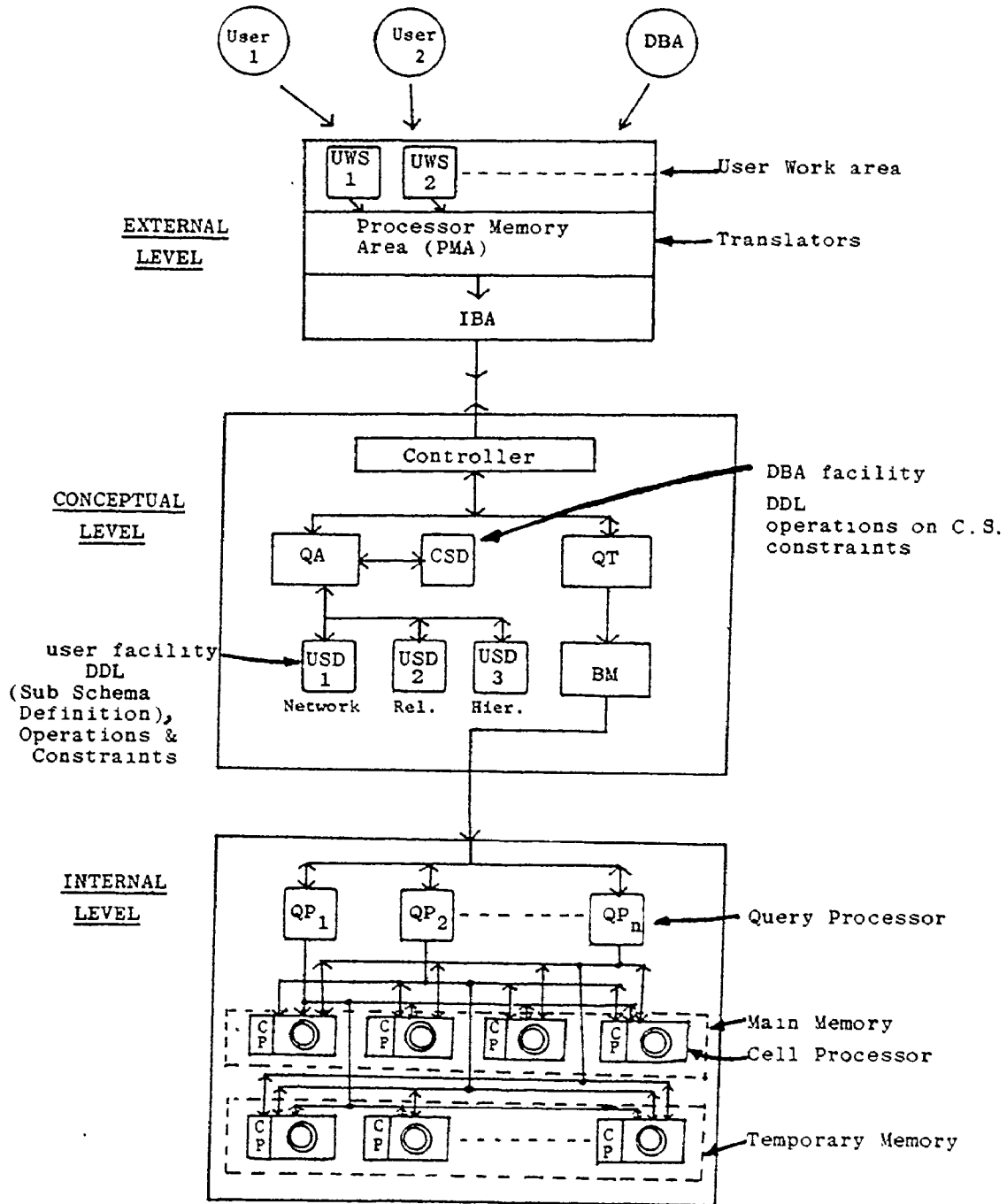


FIG. 3.1 OVERALL ARCHITECTURE OF WCRC

The IBA is organized as a first-in first-out (FIFO) memory where the queries are stored after they are translated from user languages into the conceptual level language. The translation is handled by three translators, corresponding to the three data models, which are stored as software modules in the processor memory. The jobs may also be ranked on a preassigned job priority encoding scheme.

The external level may be implemented on a host computer by software. This would involve developing program packages in the host language such as COBOL or PASCAL, to handle the translation as well as the other housekeeping activities. The details of the implementation procedures are out of the scope of this thesis.

The conceptual level supports the conceptual model and the user subschemas in the three major data models. The conceptual level model is based on the Entity-Relationship Model [CHEN 76]. The language at the conceptual schema is the Well Connected Relational Language (WCRL). [ARORA 80]. The conceptual level has the facilities for query optimization and translation from WCRL into machine language (WCRML) and a data dictionary. In addition, it also provides facilities for Data Base Administrator (DBA) to act directly on the conceptual model of the data base. A special high level language (DBAL) is developed to handle the DBA requirements.

The conceptual level consists of eight functional blocks (Fig. 3.1): The Controller, The Query Analyser (QA), The Query Translator (QT), Three User Schema Descriptors (USD's), Conceptual Schema Descriptor (CSD) and a Buffer Memory (BM). The Conceptual Processor

maintains the information about the storage structure at the internal level as well as about the conceptual model. The Query Analyser breaks up the queries into subqueries on the storage structure at the internal level. The CSD contains the information about the schema definition operations on the schema, the constraints and the data dictionary. The term 'schema' is defined as the description of a data base [CODASYL 71]. It describes the logical units of data in the data base and specifies the integrity constraints and security measures such as access restrictions to certain units of data. The description of a view by an application is called a subschema. It is a logical subset of a schema. The USD's at conceptual level contain the subschema definition, operations and constraints corresponding to views in three models. The information in CSD and USD's is used by the QA during query analysis. The QT performs the task of translating analysed queries in WCRL into machine level primitives (WCRML) that can be directly executed by the internal level and stores them in the Buffer Memory.

The internal level consists of a number of Query Processors (QP's) and an array of Cell Processors (CP's). Each QP is a master processor which is responsible for executing one query in machine language using the cells. The cells are logic per track devices. The data is stored on the cell tracks as binary partitions, known as pseudo-canonical partitions (PCP's). A formal discussion on PCP's is presented in Chapter IV, along with the description of WCRL, for the sake of continuity. For the present overall discussion, it would

suffice to note that PCP's are partitioned binary relations. All the cells are independent of each other, i.e. there is no direct communication among them. They can only communicate through the query processor which controls them. The data on the cells can be read by any number of QP's but only one QP can write on to the track at a time. The QP selects the cells required by the query it is handling and makes them slaves. They are released once the query is processed. The QP co-ordinates its slaves and also computes the overall set results. Several queries can be handled at the same time as the QP's can work in parallel. This is a multiple instruction - multiple data stream organization (MIMD).

The memory at the internal level is divided into two parts - the permanent memory stores the data base and the temporary memory stores any intermediate results of a query, if required. The data structure and hardware organization are discussed in detail in Chapter V.

### 3.2 Query Processing

In this section, an overview of how a query is processed at the three levels is discussed. The queries originate at the user end in three languages corresponding to three different models. These are translated into WCRL, priority encoded and put in a queue into the IBA by the External Processor. The system also supports the Data Base Administrator who works on the conceptual level directly but reaches it through external level like any other user. Therefore, some queries



may be originated by the DBA.

Once a query is received by the external processor, it is checked for syntax and violation of any security or integrity constraints. After it successfully goes through these stages, it is translated and put on the job queue, the conceptual processor is activated. It takes one query at a time from the IBA and performs query analysis on it. It breaks up the query on logical well-connected relations (WCR's) into subqueries on physically stored PCP's. It checks whether logical WCR's can be constructed from the stored ones or not and also subjects them to constraints such as security and integrity. Any violation of these constraints would terminate the query at this stage and an error message would be channelled to the user through the external level. The successful queries would now be translated by the Query Translator into WCRML and submitted to the internal level through buffer memory. The query translator also provides the information about the required cells along with the machine primitives. This minimizes the search time required to select the cells. The QP's take the queries from the buffer memory and select the required tracks by a polling scheme. The machine languages primitives are executed on the slave processors. For the retrieval queries, the data is sent to the external level directly through an I/O read mechanism. For the update queries, the success/failure is communicated to the user.

These various stages of query processing are illustrated in Fig. 3.2.

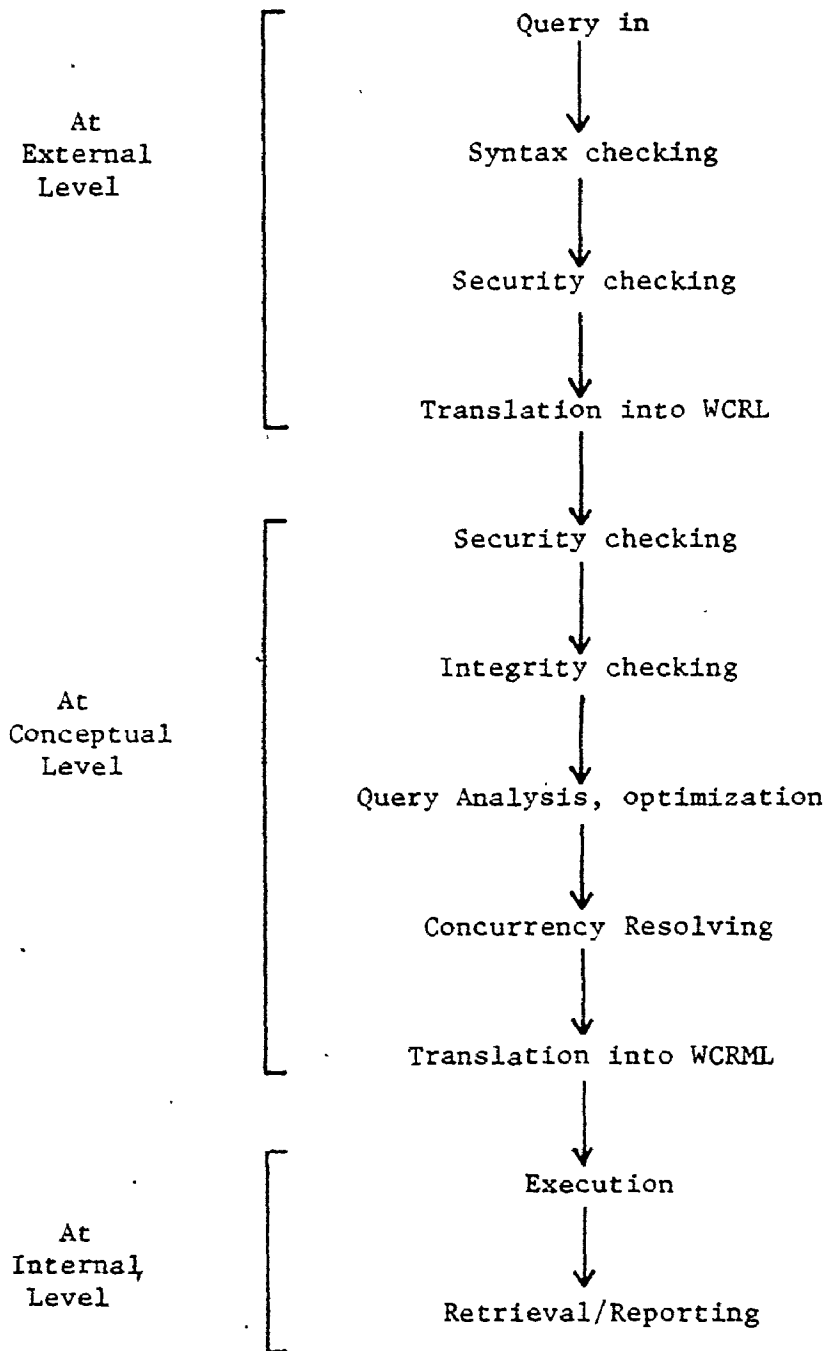


Fig. 3.2 The Stages of Query Processing

### 3.3 Data Base Facilities in WCRC

In this section, some desirable DBMS facilities provided by WCRC, for the user as well as for system operation, are examined. They include the external language facilities, facilities for the security and integrity maintenance, view generation and updates through views, concurrency and provision for data base growth. The implementation strategies for these facilities are outlined here. The detailed software implementation is out of the scope of this thesis.

#### 3.3.1 The External Languages

The WCRC supports the following query languages at the user level:

- a) SEQUEL [BOYCE 74] for the relational model,
- b) LSL [TSICHRITZIS 76] for the network model
- c) IMS Data Sublanguages [DATE 76] for the hierarchical model

These languages provide two types of facilities to the user: a data manipulation facility and a data definition facility. The corresponding subsets of a user language are called as the Data Manipulation Language (DML) and the Data Definition Language (DDL). The DML is used by the user to select the data and cause the selected data to be transferred from the data base to the application program. On the other hand, DDL allows the user to describe his view of the

data base and constraints on them. —

The views in the three models are mapped into the conceptual model by a set of transformation rules. Three translators mapping the external languages into conceptual language (WCRL) would suffice to support the external languages. The details of WCRL, the transformation rules and some examples showing the translation of queries in SEQUEL, LSL and IMS are presented in Chapter IV.

As mentioned earlier, WCRC supports the DBA at the external level like any other user. For this purpose, a high level language, DBAL is developed. The details of DBAL are presented in Chapter IV. The DBAL provides facilities for Storage Definition Language (SDL) in addition to DML and DDL. The SDL allows the DBA to describe the physical storage of the data. Thus, DBA can define and create the data base, make changes to the data base, and access the data to monitor performance.

### 3.3.2 Security and Integrity

WCRC also provides facilities for protecting the data base against unauthorized disclosure or alteration. This is implemented at two levels in the form of security constraints. Firstly, at the user level, it involves specifying a password or some other identification to invoke programs or to issue certain commands. For example, to log on, the user may be required to supply a password. Similarly, to delete, insert or modify a record(s), or a data item, the user may have to supply a code word. These checks are maintained

at the external level and implemented during translation by the external processor. Secondly, at the system level, a data dictionary is maintained which includes the security information regarding the data. The data dictionary constitutes a part of the CSD and it is common to all the three models supported by the system. The data dictionary includes the following information for all attributes at the conceptual level:

- i) Attribute name
- ii) Data type (Integer, Real, Character, etc. ...)
- iii) Entity/Relationship name in which it is participating
- iv) Synonyms, if any
- v) Security code to allow updating
- vi) User access control to attributes

For example, the access control to users may be organized as a User Access Table (UAT) in the data dictionary as shown in fig. 3.3. The table shows user privilege associated with each attribute to read, write, modify or perform statistical operations such as average and sum on the data.

The data base integrity is maintained in WCRC in the form of constraint checking to ensure that the data is always accurate. These constraints would protect the data base from any invalid or illegal alterations to the data. Integrity constraints are maintained at the conceptual level in the USD's and the CSD. They are primarily of two types:

		Attributes				
ID#	Attr.	A <sub>1</sub>				A <sub>2</sub> ----- A <sub>m</sub>
		R	W	M	S	
Users	1	✓	X	✓	✓	
	2	✓	X	X	X	
	3	✓	✓	✓	✓	
	.					
	n					

Fig. 3.3 The User Access Table (UAT)

- i) System defined constraints (by DBA)
- ii) User defined constraints

The system defined constraints are based on the conceptual schema of the data base. They are specified by the DBA using data definition language. They are invoked during query processing in the conceptual processor. The user defined constraints are based on the views of the data base. The DDL facilities in the user languages are used to specify these constraints.

The following types of integrity constraints are provided in WCRC:

- i) Specification of allowable/permitted/'unique' values for the attributes. (EX: 'salary should not exceed certain amount', 'Age should be between 20-55 years', etc. ...).
- ii) Specification of key values and functional dependencies in the conceptual model.
- iii) Specification of the type of relationships in the conceptual model (i.e., 1:1, 1:N, M:N, total, partial, etc. ...).
- iv) Consistency constraints such as "the sum of the budgets of the individual departments should be equal to the total budget of the organization". (These are specified by the DBA).
- v) Access control (as mentioned under security constraints) to prevent invalid alterations.

It may be noted that integrity constraints are closely related to the security constraints. Further, the choice of entity-relationship model at the conceptual level and the DBA language make it easier to specify these constraints. These details are presented in Chapter IV. The Query Analyser at conceptual level is responsible for invoking the constraint checking routines.

### 3.3.3 View Generation and Updates Through Views

As mentioned earlier in this chapter, WCRC allows the users to define their own views using the data definition facilities provided at the external level. Views have three primary benefits [CHAMBERLIN 75]. They simplify the user interface by allowing the user to ignore the data that is of no interest to him. They enhance the data independence, as changes in the data base need not have any impact on the views. Finally, they provide a measure of protection by preventing a user from accessing the data outside his view.

The external schemas corresponding to the conceptual model of the data base are available in all three models with the DBA. Before the user defines his view, he obtains the necessary information about the external schema of his interest, from the DBA. Unlike GDBMS [DOGAC 80], we do not provide the user with all the information, for security reasons. Once the subschemas are defined, they are translated into views in the entity-relationship model. These views are checked to find out whether they can be derived from the main schema or not. The views which are not derivable or violate



some integrity constraints will be rejected at this stage. The USD's maintain the description and constraints of the views. Once a view is 'undefined' the corresponding information is deleted from USD's.

The following criteria are suggested to be taken into consideration to define allowable views:

i) Views in the relational model may be derived using relation algebra on the 'base' relations which appear in the relational model of the main schema. Fig. 3.4 illustrates the view construction.

ii) A view in the hierarchical model should be a subset of the hierarchical trees or a subtree of one such tree in the hierarchical model of the main schema. However, the subschema may have a rearrangement of a tree or a part of the tree such that: a) The nodes in the subschema tree are a subset of nodes in the external schema, b) For each node in the subschema tree, the set of ancestor and descendent nodes in the subschema is a subset of the same in the external schema. The descendents of a node  $n$ , are those nodes in the tree, which have  $n$  as the root (including  $n$ ). The ancestors of  $n$  are all those nodes for which  $n$  is a descendent (excluding  $n$ ). Some possible views in the hierarchical model are illustrated in fig. 3.5.

iii) A view in the network model may be a subset of the record types and the links in the corresponding external schema. Sometimes, the record types in the view may be formed by join of certain record types in the schema. The user language LSL provides facilities to define such views [TSICHRITZIS 76]. Fig. 3.6 illustrates some examples

External SchemaSome Subschemas $R_1$  [EMP, DEPT]i)  $V_1$ :  $R_1$  [EMP, DEPT] $R_2$  [EMP, SAL, AGE] $R_2$  [EMP, SAL, AGE]ii)  $V_2$ : ( $R_1$  [EMP=EMP]  $R_2$ ) $R_3$  [DEPT, MGR, PROJ]

[EMP, DEPT, SAL]

Fig. 3.4 Some Views in Relational Model

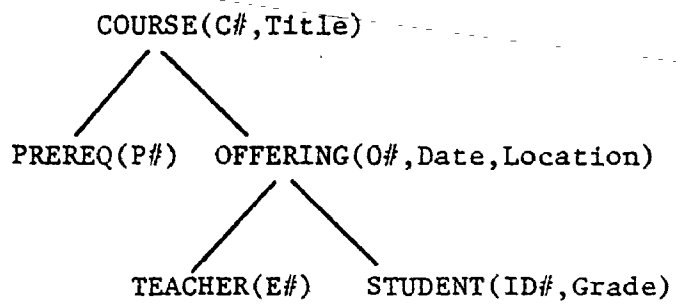
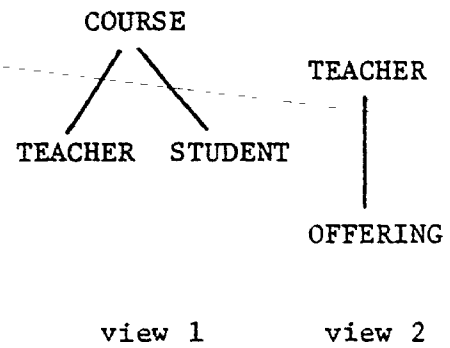
External SchemaSome Subschemas

Fig. 3.5 Some Views in Hierarchical Model

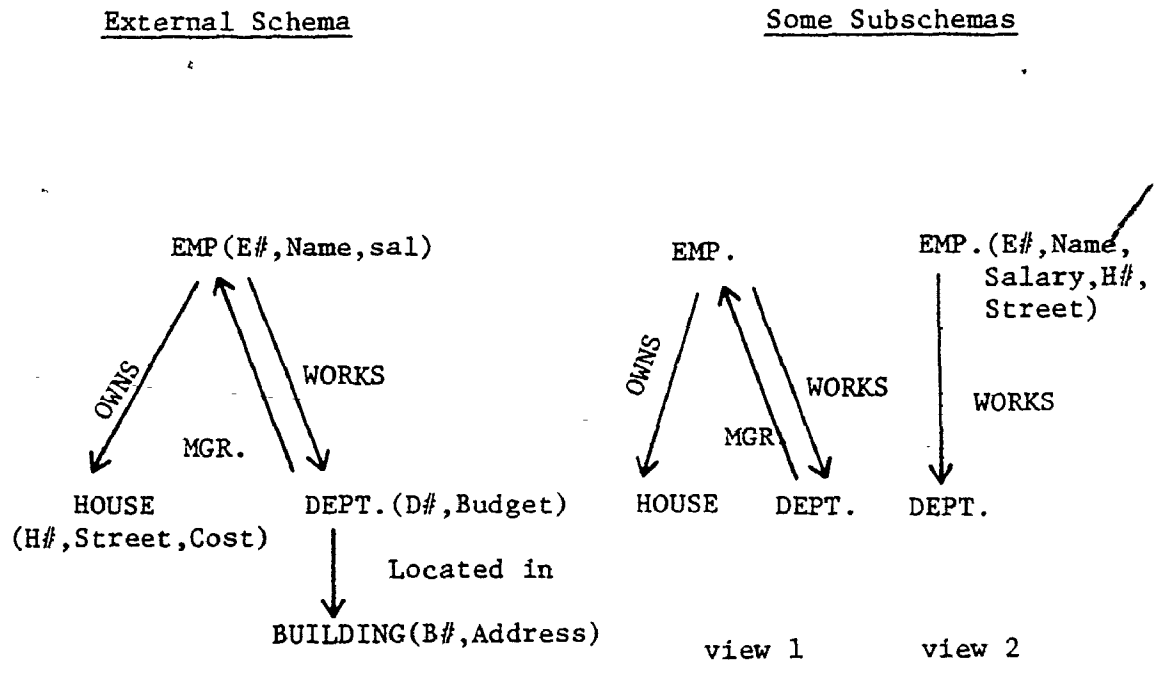


Fig. 3.6 Some Views in Network Model

of views in network model.

As views in the three models are translated into views in the conceptual model, the view operations cannot be performed directly. They must be first translated into equivalent operations on the conceptual model. After translation, only those updates which do not violate the integrity constraints will be allowed. This problem of permitting updates has been well investigated in the literature [LEWIS 78], [FURTADO 79], [OSMAN 79]. However, all these approaches address the update problem where both schema and the subschemas are in the relational model. In our case, the main schema is in Entity-Relationship model and the subschemas are in three different models. Therefore, the update strategy is invoked at the conceptual level. This problem is revisited in Chapter IV in the context of conceptual model.

As far as the external level is concerned, the following types of updates are allowed in the views:

- i) Insertion, deletion and modification of tuples in the relational model.
- ii) Insertion, deletion and modification of record occurrences and links in the network model [TSICHRITZIS 76].
- iii) Insertion, deletion and modification of record occurrences in the hierarchical model [DATE 76].

#### 3.3.4 Concurrency

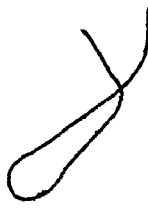
Since WCRC supports multiple queries simultaneously, a strategy is called for, to allow only those concurrent operations that preserve integrity. There are two types of concurrent operations in WCRC:

- i) Inter-query concurrency,
- ii) Intra-query concurrency.

The inter-query concurrency refers to execution of several queries simultaneously while the intra-query concurrency corresponds to parallel execution of the same query on several cells at the internal level. We follow the following conflict-free strategy for both the types. If two adjacent tasks (queries or subqueries) in a sequence do not have any data shared in common, they are executed in parallel. Otherwise, the tasks are serialized. Concurrent updates on the same data are not supported by the system. Further investigation is needed for a more sophisticated concurrency policy.

#### 3.3.5 Data Base Growth

Every data base is likely to increase in size with time. Therefore, a DBMS should make provisions in the design to accommodate data base growth. In WCRC, there are provisions at two levels to handle growth. Firstly, at the internal level, the organization of cells and query processors is modular. This makes it possible to expand storage as well as the processing capacities of the system



by simply adding more modules. The second, at the conceptual level, the DBA is provided with language facilities to change the conceptual schema. These facilities include adding, deleting, shifting and renaming of the entities, relationships and the attributes of the conceptual schema.

In the discussion so far, the main units of the system and the DBMS facilities provided have been introduced. In the next chapter, the features of conceptual level are described in detail.

## CHAPTER IV

### THE CONCEPTUAL LEVEL

#### 4.1 Introduction

As mentioned in the previous chapter, the conceptual schema represents a common view of data that encompasses all applications. All the external schemas are mapped into the conceptual schema, which has an overall view of the logical data organization of the 'real world' or the 'enterprise'. Some of the criteria that should be taken into consideration during the design of the conceptual level, are the following:

- i) The data model and the language at the conceptual level should preserve a high degree of data independence.
- ii) The model should be flexible enough to be easily translated into the external data models.
- iii) The software involved to implement the conceptual level should be of minimal complexity to improve the reliability of the system.
- iv) The response time degradation due to conceptual level should be as small as possible in order to achieve high performance in a real time environment.



#### 4.2 Choice of the Data Model

A number of data models have been proposed in the literature, out of which some have been considered for the conceptual level. The most widely used data models include: the network, the hierarchical, the relational, the entity set [SENKO 73] and the entity relationship models [CHEN 76]. The first four models are not very suitable for the conceptual level for the following reasons.

- i) The hierarchical model does not permit enough flexibility for a conceptual model. It is not always possible to represent efficiently all types of relationships among data in a hierarchical organization [KLUG 78].
- ii) Though network model is more flexible, its capability to achieve data independence is limited [CODD 70].
- iii) The relational model provides flexibility as well as a high degree of data independence. But it cannot truly represent the semantic information in the real world data. For instance, ability to perform arbitrary joins may give rise to semantic problems [SCHMID 75].
- iv) The entity set model uses only binary relationships and provides a high degree of data independence. However, it identifies values with entities, which makes it difficult to capture natural view of the real data [CHEN 76].

On the other hand, the entity-relationship model possesses the following characteristics which make it appropriate for the con-

ceptual model.

- i) It adopts closely the natural view of the real world.
- ii) It incorporates semantic information about the data.
- iii) It provides high degree of data independence.
- iv) Other data models are easily derivable from it.
- v) It avoids many problems associated with normalization. It always achieves 3NF at least.
- vi) It allows for changes in the schema and thereby provides facility to handle data base growth.

In view of these reasons, the entity-relationship model has been chosen as the conceptual model in WCRC. A review of the model is presented in the next section to facilitate further discussion.

#### 4.2.1 The Entity-Relationship Model (E-R Model)

There are four basic objects in this model: entities, relationships, attributes and value sets.

Definition 4.1: An entity is a thing which can be distinctly identified during design of a data base.

EX: Employee, Company, House, etc. ... A set of entities with common properties constitute an entity set.

Definition 4.2: A relationship is an association among entities.

EX: 'Work' is a relationship between employee and company. In general, it could be binary 1:1, 1:N, N:M type of association or a k-ary type of relationship. A set of same type of relationships

among entities is called a relationship set.

Definition 4.3: The value sets describe the information about an entity or a relationship.

EX: 'No.of years' is a value set that describes age of the entity 'Employee'.

Definition 4.4: An attribute is defined as a function which maps from an entity or relationship set into a value set or Cartesian product of value sets.

EX: 'AGE' maps the entity set employee into value set 'no. of years'.

In addition, the function performed by an entity in a relationship may be specified by "role". EX: The relationship among employees may have roles, 'Manager' and 'Employee'.

An entity is identified by a primary key among its attributes, and a relationship by combination of primary keys of the entities involved. Sometimes, if an entity cannot be uniquely identified by its own attributes, a relationship is used for its identification. Such a weak entity is identified by one of its attributes and primary key of the entity supporting it through the relationship. Thus entities and relationships could be logically viewed as relations with the attribute names and role names forming the intension (i.e. the description) and the Cartesian product of value sets forming the extension (i.e. the tuples). Certain integrity and consistency constraints may be specified on the value sets, such as allowable values, permitted values, and relationships such as 1:N relationship.

The model also allows for changes due to the data base growth. Entities and relationships constitute the upper domain, while attributes and value sets form the lower domain of the model. The following operations are employed to effect the changes within the domains: 'Split', 'Merge', 'Add', 'Delete'. EX: Adding an entity to the existing model. In addition, 'Shift' operation is used to carry out changes between the domains. EX: Changing a value set into an entity. An example of a conceptual schema expressed as an Entity-Relationship (E-R) diagram is shown in Fig. 4.1 (A corporate data base).

#### 4.2.2 Derivation of Other Data Models

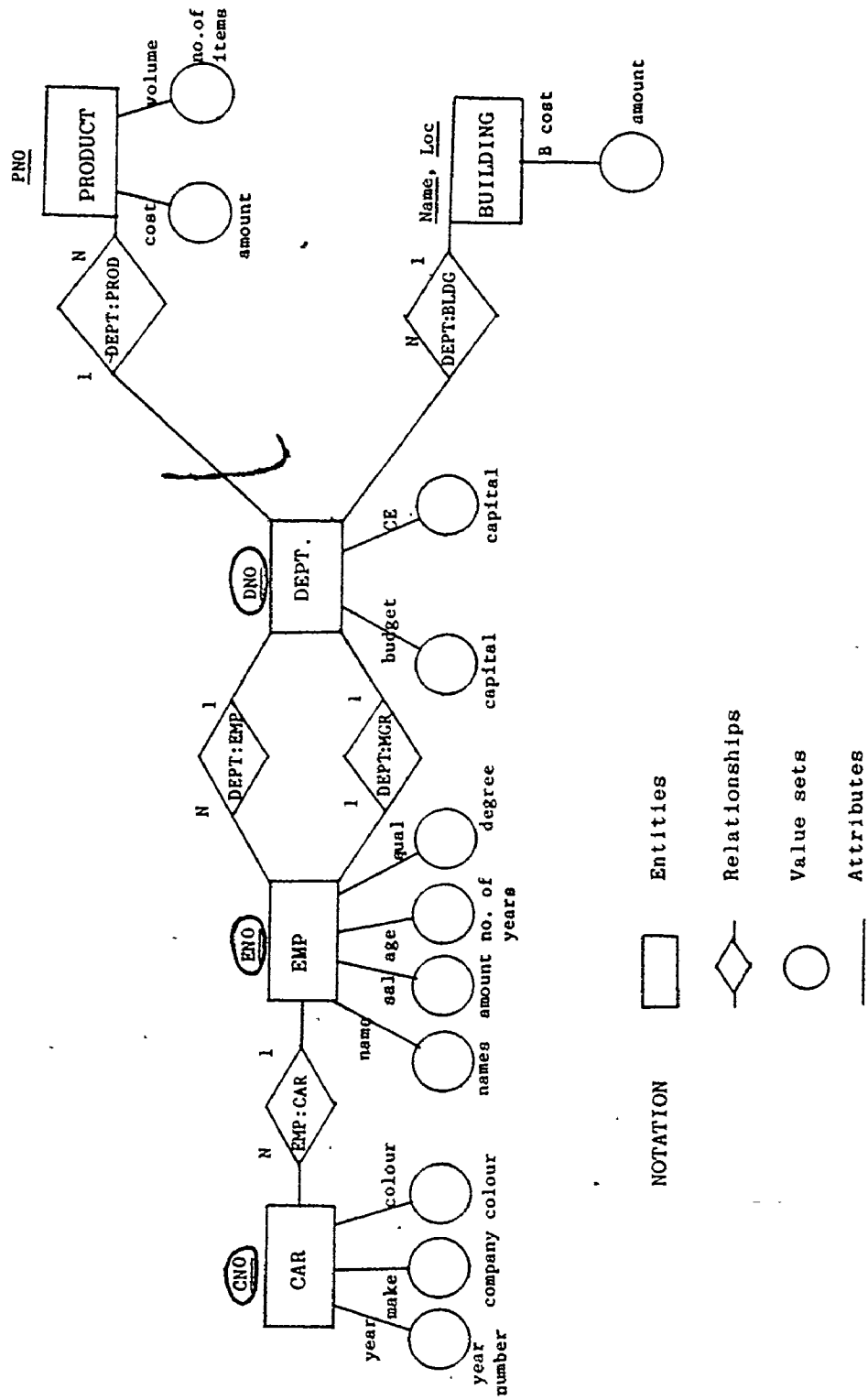
The relational model is easily derivable from the E-R model. The entity and the relationship relations correspond to 3NF relations in the relational model. The semantic information of functional dependencies (FD's) is maintained as FD's among the entity and the relationship relations (ER's and RR's). The corresponding relational model is free of many semantic problems addressed by the normalization theory.

A meaningful relational model may be obtained from the E-R model by the following conversion algorithm.

##### Algorithm for E-R Model to Relational Model Conversion

- Step 1: For each entity relation, define a corresponding relation in the relational model.
- Step 2: For each k-ary relationship, generate a corresponding relation.

FIG. 4.1 E.R. DIAGRAM OF CORPORATE DATABASE



- Step 3: For each binary relationship with attributes, generate a relation in the relational model.
- Step 4: For each binary relationship without attributes, generate a relation only if it is of the type N:M. If it is of the type 1:N or 1:1, associate the key of the source entity as an attribute with the relation corresponding to the target entity. (We define the source and target entities as follows. Let  $E_1$  and  $E_2$  be the entities involved in a relationship R, of type 1:N. Then  $E_1$  is referred to as the source entity and  $E_2$ , the target entity. Note, for 1:1 type relationship, we choose to call  $E_1$  as the source entity).

The relations obtained by this algorithm would always be in 3NF because each relation would have a primary key on which all the other non-key attributes would be functionally dependent. An example illustrating the relational equivalent of the E-R model of Fig. 4.1 is shown in Fig. 4.2.

The network model can be derived from E-R model by treating entity relations as record types and relationship relations as links. The data structure diagram [BACHMAN 69] representation of the network model can be obtained from E-R model by the following algorithm.

Algorithm: E-R Diagram to Data Structure Diagram Conversion

- Step 1: Represent every entity and relationship relation as a record type in data structure diagram.



- Step 2: For each 1:N binary relationship draw unidirectional arrows in the diagram from source entity to the target entity. For 1:1 relationship, draw a bidirectional arrow.
- Step 3: For each N:M relationship, create a new record and draw pointed arrows from the entities involved. The same holds good for k-ary relationships ( $k > 2$ ). As an illustration, the output of the algorithm for the example in Fig. 4.1 is shown in Fig. 4.3.

The following algorithm converts an E-R model into an equivalent hierarchical model.

#### Algorithm to Convert E-R Model into Hierarchical Model

- Step 1: Convert all 1:N relationships into a one-level tree with source entity as the root. As a special case, for 1:1 relationships, associate the information carried by the relationship with the node corresponding to source entity.
- Step 2: Convert N:M relationships into two trees. The parent node in the first tree would be the child node in the second tree and vice versa. The child nodes are of course renamed.
- Step 3: Convert a k-ary relationship into a one-level tree with a newly defined node as the root and the k nodes as leaves. The new node is a record consisting of the keys of the k nodes and attributes of the relationship.
- Step 4: Combine all the trees obtained above at the common nodes. This would give rise to a forest of hierarchical trees.



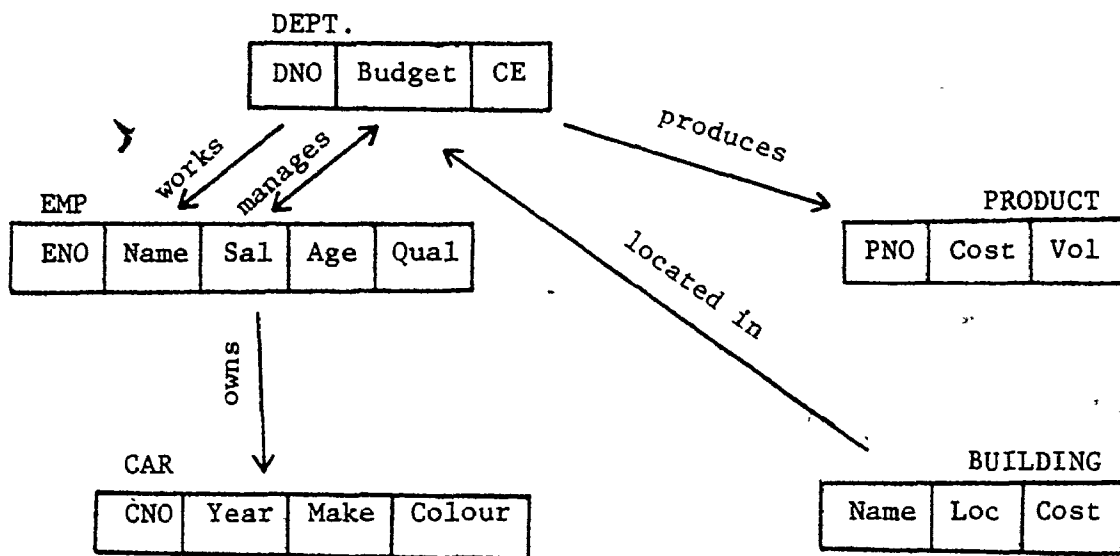


Fig. 4.3 Network Equivalent of the Corporate Data Base

The above algorithm yields the hierarchical model shown in Fig. 4.4 for the example shown in Fig. 4.1, Step 2 for N:M relationships is illustrated in Fig. 4.5.

These algorithms provide the DBA with the external schemas corresponding to the conceptual schema. Users may obtain this information from the DBA while defining their views.

#### 4.3 Choice of the Language

As mentioned earlier, WCRC supports several user languages at the external level. This makes it necessary to have a common data model independent language at the conceptual level. Some of the data model independent languages proposed in the literature are LSL [TSICHRITZIS 76], FQL [BUNEMAN 79], QUEST [HOUSEL 79] and WCRL [ARORA 80]. We use WCRL as the language for conceptual level of WCRC for the following reasons.

- i) The language is based on a data structure called elementary well-connected relation (EWCR), which can be easily mapped into the storage structure at the internal level, a binary pseudo-canonical partition (explained later in detail).
- ii) It allows complex conditional expressions involving set comparators, aggregate functions, and recursion, some of which are not available in the other languages.
- iii) It is algebraic in nature and also allows navigation through the model.

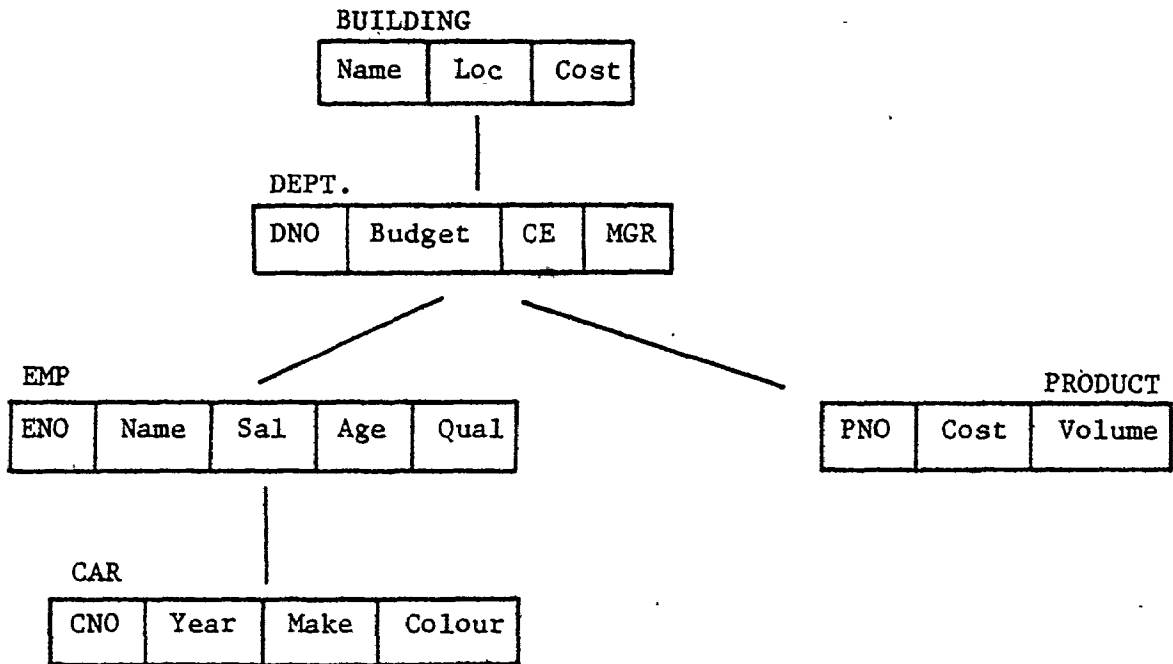


Fig. 4.4 Hierarchical Equivalent to the Corporate Data Base

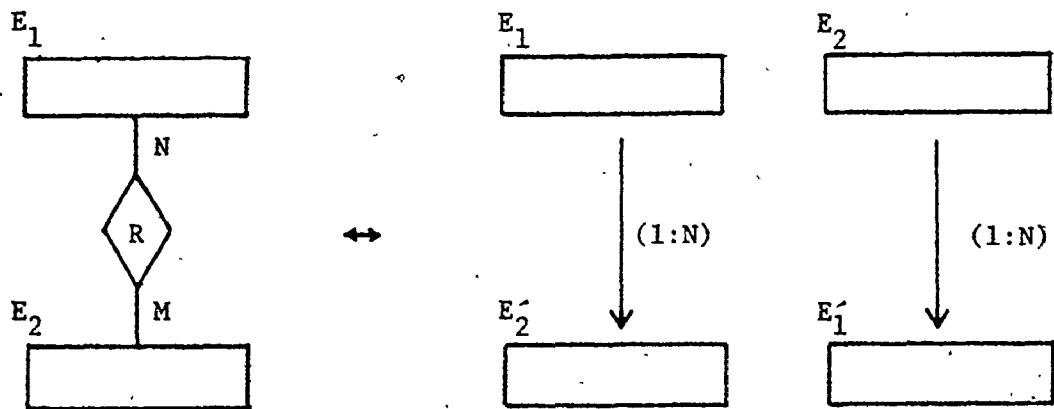


Fig. 4.5 Hierarchical Tree Corresponding to a N:M Relationship

- iv) It applies equally well to the three major data models - Network, relational and hierarchical.

The language as described in [ARORA 80] provides only data retrieval facilities. In this report, we extend the language to accommodate data definition, data manipulation (update) and storage definition. In the sections which follow, these aspects are discussed in detail.

#### 4.3.1 WCRL - Data Retrieval Language

WCRL is based on the theory of well-connected relations [ARORA 79]. Those definitions and the language commands which are relevant for the discussion, are presented below.

Definition 4.5: A Well-Connected Relation (WCR) is a binary relation  $W$ , on two sets  $A$  and  $B$  such that

$$(\forall a) (a \in A) (\forall b) (b \in B) (aWb)$$

The sets  $A$  and  $B$  are called the first and the second constituents of the WCR. Two WCR's are compatible if their constituents are based on the same domains.

Definition 4.6: An Elementary Well-Connected Relation

(EWCR) is a WCR in which the first constituent has a single element. The second constituent is then called the 'Image set' of the first constituent.

Definition 4.7: A Trivial Well-Connected Relation (TWCR)

is a WCR in which both the constituents have a single element.

Definition 4.8: A relation  $R[A, B]$  can be expressed as

$$\begin{aligned} R[A, B] &= \sum_{i=1}^n R_i[A_i, B_i] \\ &= R_1[A_1, B_1] \cup R_2[A_2, B_2] \cup \dots \cup R_n[A_n, B_n] \\ &= \pi(R), \text{ a partition of } R \end{aligned}$$

$$\text{where } R_i[A_i, B_i] \cap R_j[A_j, B_j] = \emptyset$$

$$\text{for } i \neq j, i \geq 1, j \leq n \text{ and } A = \bigcup_{i=1}^n A_i \text{ and } B = \bigcup_{i=1}^n B_i$$

Definition 4.9: A partition of a binary relation  $R[A, B]$  is a canonical partition (CP) if,

$$R[A, B] = \sum_{i=1}^n w_i[A_i; B_i]$$

where, i)  $w_i[A_i; B_i]$  is a WCR for  $1 \leq i \leq n$ ,

ii)  $A_i$  is a set with a single element for  $1 \leq i \leq n$ ,

iii)  $A_i \neq A_j$  for  $i \neq j$  and  $1 \leq i, j \leq n$

A graphical illustration of these definitions is shown in Fig.

4.6.

WCRL provides two levels of operations based on set processing. This means, any condition on the first constituent of an EWCR

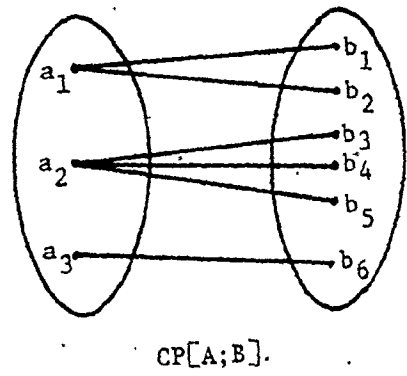
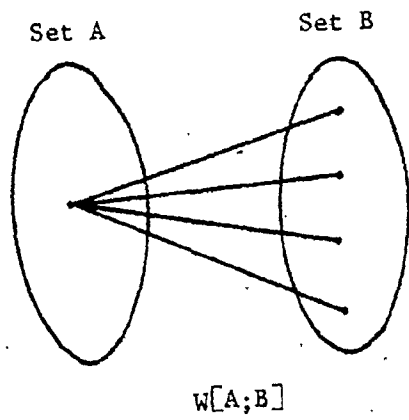
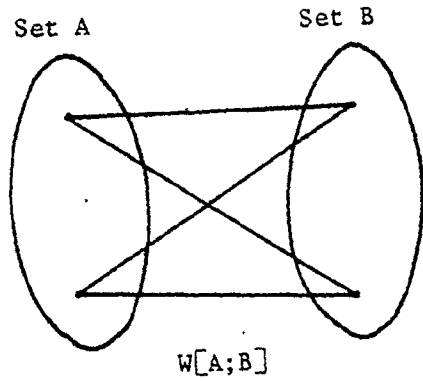


Fig. 4.6 Illustration of a WCR, an EWCR and a CP

is evaluated only once for all the tuples in the EWCR. The two levels in WCRL are the low level and the high level. The former uses only EWCR's as operands while the latter uses canonical partition. The operations fall into three classes, namely, set reconstitution, set join and pseudo set operations. These operations are briefly discussed below.

i) Set Reconstitution ( $\sigma_{sr}$ ):

The algebraic representation of this operation is given by,

$$\sigma_{sr} W [A; B] (\text{First Expression } (x); \text{ Second Expression } (y)) = C [X; Y]$$

where expression may be,

- a) A list of attributes belonging to (A U B)
- b) Null
- c) A list of attributes where some attributes are restricted or selected according to a relationship.

A qualification may involve arithmetic (+, -, \*, ÷) operators and aggregate functions. For example, X may be of the form,

$$\underline{\text{EX:}} \quad X = A_1, (B_1 = 1000), (B_2 = \text{AVG } (B_3) \div 100)$$

The result of this operation is a CP, C(X; Y).

ii) Set Join ( $\sigma_{sj}$ ):

It is of the form,

$$\sigma_{sj} w_1 [A_1; B_1], w_2 [A_2; B_2], \dots, w_n [A_n; B_n] (\text{conditional expression}) = C [X; Y]$$

where  $X = A_1, A_2, \dots, A_n$

and  $Y = B_1, B_2, \dots, B_n$

Codd's join ( $\sigma_{cj}$ )



or  $X = A_1, B_1, A_2, B_2, \dots, A_n$

Brooming Join ( $\sigma_{bj}$ )

and  $Y = B_n$

The conditional expression is a set of conditions connected by boolean operators ( $\wedge, \vee, \neg$ ). Each condition may involve relational operators ( $=, >, \geq$ , etc.) set comparison operators (current  $A_1 > \text{All } B_2, \text{All } A_1 \leq \text{some } B_1$ , etc.) and set containment operators ( $A_1 \subseteq B_1, A_2 \not\subseteq B_2$ , etc.) on any two attributes from A's and B's.

EX: A conditional expression may be of the form,

$$(A_1 = 50) \wedge (A_2 \subseteq A_3) \wedge (B_2 > 100) \vee (B_2 \cdot \geq \dots B_3)$$

where ' $\cdot \geq \dots$ ' stands for some  $B_2$  is greater than or equal to all  $B_3$ .

(Note, 'one dot' corresponds to 'some', 'two dots' to 'all' and 'no dot' to 'current').

### iii) Pseudo Set Operations:

These operations are 'selective union', 'selective intersection' and 'selective difference'. The arguments of these operations should all be compatible EWCR's.



a) Selective Union ( $\sigma_{su}$ )

$\sigma_{su} w_1 [A; B_1], w_2 [A_2; B_2], \dots, w_n [A_n; B_n]$  (conditional expression) =  $w [A; B]$

b) Selective Intersection ( $\sigma_{si}$ )

$\sigma_{si} w_1 [A_1; B_1], w_2 [A_2; B_2], \dots, w_n [A_n; B_n]$  (conditional expression) =  $w [A; B]$

c) Selective Difference ( $\sigma_{sd}$ )

$\sigma_{sd} w_1 [A_1, B_1], w_2 [A_2, B_2], \dots, w_n [A_n, B_n]$  (conditional expression) =  $w [A; B]$

In selective union, the tuples from the EWCR's that satisfy the conditional expression are unionized into one EWCR. Selective intersection collects the common tuples from the EWCR's that satisfy the expression and forms a new EWCR. Selective difference removes the common tuples from  $w_1$  and retains only those that satisfy the expression in  $w_1$ .

The high level operations are similar to the low level ones except that the arguments are canonical partitions instead of EWCR's. When CP's are involved, the low level operation is applied to each EWCR/set of EWCR's in the CP.

In order to demonstrate the suitability of WCRL as a concep-

tual language, the language must be further developed. The operations for data definition, data manipulation (for updates) and storage definition are developed in sections 4.3.2, 4.3.3 and 4.3.4.

#### 4.3.2 Data Definition Language (DDL) for WCRL

The DDL of WCRL presented here, is intended for defining the schema and the constraints of the entity-relationship model. The various statements are given below.

##### i) Entity Definition Statement:

This statement takes the form,

$$\sigma_{de} \ a_1, a_2, \dots, a_n \ (\langle E\_SPECIFICATION \rangle) = e(E_i)$$

where a)  $a_1, a_2, \dots, a_n$  represent the attributes of the entity,

b)  $E_i$  is the name of the entity,

and c)  $\langle E\_specification \rangle$  is of the form  $(t_e, K, R)$

where  $t_e$  indicates the type of the entity (weak, normal),

K denotes the key and R specifies the supporting relationship in the case of a weak entity.

EX:  $\sigma_{de} \ \text{DNO, Budget, CE} \ (\text{Normal, DNO, } \phi) = e(\text{Dept.})$

##### ii) Relationship Definition Statement:

The algebraic representation of this statement is given by,

$$\sigma_{dr} \ a_1, a_2, \dots, a_n \ (\langle R\_SPECIFICATION \rangle) = r(R_i)$$

- where
- a)  $a_1, a_2, \dots, a_n$  are the attributes of the relationship,
  - b)  $R_i$  is the name of the relationship,
- and
- c)  $\langle R\_specification \rangle$  is of the form  $(d, \langle spec. list \rangle)$

Here  $d$  indicates the degree of the relationship (i.e. binary,  $k$ -ary), and the  $\langle spec. list \rangle$  constitutes  $(s_1, s_2, \dots, s_i, \dots, s_k)$  where each  $s_i$  is of the form  $(e_1, e_2, t_r)$ ;  $e_1$  and  $e_2$  denote the entities involved in a relationship while  $t_r$  indicates the type (i.e. 1:N, 1:1 or N:m). Further, the existence of some entity instances that do not participate in the relationship may be indicated by a ' $\sim$ ' sign on top of the corresponding entry in  $t_r$ . Such a relationship is called partial relationship. They are made total by inclusion of null values.

EX: To define N:M relationship 'Work' between entities 'Employee' and 'Project' where every employee is not required to work on at least one project, the following statement may be used.

$$\sigma_{dr} \text{ No. of Hours (binary, (Employee, Project, } \overset{\sim}{N:M}) \\ = r(\text{work})$$

iii) Attribute Definition Statement:

This statement takes the form

$$\sigma_{da} \vee (\langle V\_SPECIFICATION \rangle) = a(A_i)$$

where a)  $v$  denotes the value set that forms the domain of the attribute,

b)  $A_i$  is the name of the attribute,

and c)  $\langle V\_Specification \rangle$  is of the form  $(t_d, u, p)$

where  $t_d$  represents the data type of the value set

(real, integer, etc.),

$u$  specifies the units (dollars, years, etc.), and

$p$  denotes the range of allowable values or uniqueness

EX: The attribute 'Salary' may be defined as follows:

$\sigma_{da}$  Amount (Integer, Dollars, 10,000 - 60,000)

= a(salary)

iv) Constraint Specification Statements:

The constraints on E-R model may be specified using the following security and integrity statements.

a)  $\sigma_{SC} a_1, a_2, \dots, a_n (SE_1, SE_2, \dots, SE_n)$

where a)  $a_i$  is an attribute or an entity

and b)  $SE_i$  is a security constraint expression on  $a_i$ . In general,

$SE_i$  is a list of expressions of the form  $(U#, \langle A\ LIST \rangle)$

where  $U\#$  indicates the user identification number and  $A\ LIST$  consists of the allowable accesses (Read(R), Write(W), Modify

(M), Read statistical values (S) such as 'Average', etc.). Thus, several constraints can be specified in one statement.

b)  $\sigma_{IC} O_1, O_2, \dots, O_n (IE_1, IE_2, \dots, IE_n)$

where a)  $O_i$  is an object name, namely, an entity, a relationship, an attribute or a value set name.

and b)  $IE_i$  is an integrity constraint on  $O_i$  of the form,

<Agr\_fn> <object name> <opr> <opd> where,

<Agr\_fn> := AVG|SUM|COUNT|MIN|MAX| $\phi$ (optional),

<object name> := entity name|relationship name|value set name|attribute name,

<opr> := set containment operator|comparison operator,

<opd> := constant|<Agr fn> (<object name>).

EX: i) The user access to an attribute 'Salary' can be specified as

$\sigma_{SC} \text{ Salary } ((U_1, (R, W)), (U_2, (M, S)))$ .

This means that user ' $U_1$ ' is allowed to read and write while user  $U_2$  is permitted only to modify and read the statistical results. Of course, modify implies read and write as well but not vice versa.

ii) The integrity constraint, "Manager's salary should be greater than average salary of the employees" can be specified as,

$\sigma_{IC} \text{ Mgr} \cdot \text{Sal } (\text{Mgr} \cdot \text{Sal} > \text{AVG} (\text{EMP} \cdot \text{Sal}))$

iii) The constraint "The managers should be a subset of employees" may be specified as,

$$\sigma_{IC} \text{ Mgr } (\text{Mgr} \subseteq \text{Emp})$$

iv) Schema Change Specification Statement:

A change in the objects of the schema may be effected by the following statement,

$$\sigma_{CS} \ o_1; o_2; \dots, o_n \ [ \langle \text{object type list} \rangle ] \ ( \langle \text{condition list} \rangle ) \\ = O(O_1; O_2; \dots; O_n)$$

where a)  $o_i$  is an object name list or an object name

b)  $\langle \text{object type} \rangle$  is either an entity, a relationship, an attribute or a value set; the  $i$ th object type corresponds to  $o_i$

c)  $\langle \text{condition} \rangle$  is of the form

$\langle \text{object name} \rangle \langle \text{opr} \rangle \langle \text{constant} \rangle$  and

$\langle \text{opr} \rangle : = = | \neq | > | < | \geq | \leq$

$\langle \text{constant} \rangle : = \text{Integer} | \text{Real} | \text{character string},$

and d)  $O_i$  is a list containing the new name(s) of the object  $O_i$ .

If  $O_i$  contains two names, the instances satisfying the condition are given the first name. The rest are given the second name.

Note, all the schema change statements of the DBA Language, split, merge, shift, rename, and delete, (explained later in section 4.4.2) can be expressed by  $\sigma_{CS}$  statement. The

other statement 'add' is taken care of by the data definition statements i, ii, iii and iv of this section.

EX: i) Change the entity 'Persons' into two entities called 'Male persons' and 'Female persons'; change the attributes

$$\begin{aligned} \sigma_{CS} \{ \text{Person; (home phone, office phone)} \} [ \text{Entity; attribute} ] \\ (\text{Person} \cdot \text{Sex} = \text{'Male'}; \phi) \\ = 0 \{ (\text{Male person, Female person}): (\text{phone}) \} \end{aligned}$$

ii) Delete attribute 'Age' of the entity person

$$\sigma_{CS} \text{ person} \cdot \text{Age} [ \text{Attribute} ] (\phi) = 0(\phi)$$

#### 4.3.3 Data Manipulation Language (DML) for WCRL

The DML of WCRL for updates, based on the entity-relationship model, comprises the following statements.

a) Insertion:

Insertion of a tuple into entity or a relationship is accomplished by the following statement,

$$\sigma_I A_1, A_2, \dots, A_n [ \langle \text{object name} \rangle ] (t_1; t_2; \dots; t_m) = 0(O_i),$$

$$\text{or } \sigma_I A_1, A_2, \dots, A_n [ \langle \text{object name} \rangle ] (\langle \text{Expression list} \rangle) = 0(O_i)$$

where a)  $A_i$  is a list of some or all attributes of the object type  
(unspecified values are taken as null values)

b)  $\langle \text{object name} \rangle$  is either an entity or a relationship name

c)  $t_i$  denotes a tuple which is of the form  $(a_1, a_2, \dots, a_n)$

where  $a_i$  is a value

d) <Expression list> gives a tuple or a set of tuples, as follows,

<Expression list>: = <expression>; <expression list> |

<expression>

<expression>: = <iexpr>, <iexpr list> | <iexpr>

<iexpr>: = <aggr func> (<atname>) |

<aggr func> (<atname>) <op> <constant>

where <atname> is an attribute name

<aggr func> is one of (MIN, MAX, SUM, AVG, CAR)

and <op> is one of (+, -, ×, ÷).

e)  $O_i$  is the 'rename' of the object type (Renaming is optional).

b) Deletion:

The following statements can be used for deleting the tuples from an entity or a relationship

$\sigma_D A_1, A_2, \dots, A_n [ \text{<object name>} ] (t_1; t_2; \dots; t_m) = O(O_i)$

or

$\sigma_D A_1, A_2, \dots, A_n [ \text{<object name>} ] ( \text{<rexpression list>} ) = O(O_i),$

where <rexpression list> is as follows

<rexpression list>: = <rexpression> <rexpression list> |

<rexpression>

<rexpression>: = <rexpr>, <rexpr list> | <rexpr>

<aggr func>



$$\langle \text{rexpr} \rangle := \langle \text{aggr func} \rangle \langle \text{atname} \rangle \langle \text{op} \rangle \langle \text{constant} \rangle$$

$$| \langle \text{atname} \rangle \langle \text{cop} \rangle \langle \text{aggr func} \rangle \langle \text{atname} \rangle,$$

where  $\langle \text{atname} \rangle$  is an attribute name,

$\langle \text{op} \rangle$  is one of (+, -, ×, ÷)

$\langle \text{cop} \rangle$  is one of (=, ≠, >, <, ≥, ≤)

and  $\langle \text{aggr func} \rangle$  is one of (SUM, AVG, MAX, MIN, CAR) .

The syntax is similar to that of insertion statement.

c) Modification:

The statements to modify tuples in entities or relationships take the form,

$$\sigma_M A_1, A_2, \dots, A_n [\langle \text{object name} \rangle] ((a_1, a_2, \dots, a_n), (b_1, b_2, \dots, b_n)) = 0(0_i)$$

or  $\sigma_M A_1, A_2, \dots, A_n [\langle \text{object name} \rangle](\langle \text{mexpression list} \rangle)$

$$(b_1, b_2, \dots, b_n) = 0(0_i)$$

where a)  $a_i$ 's are the values of  $A_i$ 's before update

b)  $b_i$ 's are the values of  $A_i$ 's after update

and c) The value for  $\langle \text{mexpression list} \rangle$  is a tuple or a set of tuples. The syntax of  $\langle \text{mexpression list} \rangle$  is same as  $\langle \text{rexpression list} \rangle$ . Note, all the three update statements provide an option to rename the object after update.

Note: WCRL also allows for recursion which can be used to define new operations based on the other retrieval operations. The detailed syntax of a recursion statement is shown in Appendix A.

#### 4.3.4 Storage Definition Language (SDL) for WCRL

The SDL part specifies the mapping between the logical and physical organization of the data (i.e. the conceptual and the internal levels). It also provides facilities for formatting, creating, destroying and renaming of the internal level storage structures. Before developing these language statements, a discussion on the mapping between conceptual and internal levels is presented here.

It may be recalled that the logical objects at the conceptual level are entities, relationships, attributes and value sets while the storage structures at the internal level are canonical partitions (CP's). There exists a simple mapping between the objects of E-R model and the CP's, as follows.

- i) As mentioned in section 4.2.1, the attributes are functional mappings from entity sets and relationship sets into value sets. Therefore, a binary relation consisting of the key of an entity or a relationship set and a value set would conveniently form a CP representing an attribute.
- ii) All binary relationships which are 1:1 or 1:N type directly correspond to the respective CP's.

iii) Any N:M type binary relationship may be transformed into a set of two CP's by associating a system defined key with it. For example, a relationship,  $R[A B]$  would become CP's,  $[K;A]$  and  $C_2 [K;B]$  where  $K$  is the system defined key. On similar grounds, a  $k$ -ary relationship may also be transformed into a set of CP's.

These mapping rules would form the basis for the following SDL statements.

a) Mapping Statements

$$i) \sigma_E c_1, c_2, c_3, \dots, c_n (\langle M\_Expression \rangle) = e(E_i)$$

$$ii) \sigma_R c_1, c_2, c_3, \dots, c_n (\langle M\_Expression \rangle) = e(R_i)$$

These statements define the entity|relationship sets respectively, in terms of the CP's.  $\langle M\_Expression \rangle$  stands for a Codd's join statement. The  $c_i$ 's denote the canonical partitions and  $E_i/R_i$  the name of entity|relationship set.

The language also allows for "logical CP's" to be defined in terms of the CP's at the internal level, by the following statement. A logical CP may have more than two attributes:

$$iii) \sigma_{CP} c_1, c_2, \dots, c_n (\langle M\_expression \rangle) = cp(C_i)$$

where  $c_i$ 's denote the CP's or other logical CP's already defined.

$C_i$  denotes the logical CP

and  $\langle M\_Expression \rangle$  stands for a set join or a pseudo set operation.

For example consider the CP's,  $C_1$  [ENAME;SAL] and  $C_2$  [ENAME;SAL] where  $C_1$  corresponds to all the employees and  $C_2$  corresponds to only managers. Suppose we need only those employees who are not managers but earn more than \$30,000. Then a logical CP,  $C_3$  can be defined as

$$\begin{aligned} \sigma_{CP} C_1, C_2 (\sigma_{sd} C_1[ENAME;SAL], C_2[ENAME;SAL] (C_1 \cdot SAL > 30,000)) \\ = cp (C_3[ENAME;SAL]) \end{aligned}$$

Note, These statements may involve some data retrieval statements such as set join in their <M\_expression> part. The same statements may also be used for renaming by letting M\_expression to be null.

#### b) Storage Definition Statements

##### i) Format:

The following statement is used to specify the format of the canonical partitions,

$$\sigma_{FM} c_1, c_2, \dots, c_n (<F_1, F_2, \dots, F_n>)$$

where  $c_i$ 's are the CP's and  $F_i$  is of the form  $<f_1, f_2>$ .  $f_1$

represents the first constituent and  $f_2$ , the second constituent.

Each  $f_i$  is a 3-tuple of the form  $<n, t_d, n_b>$  where  $n$  is the name of attribute,  $t_d$  the data type and  $n_b$  the number of bytes.

EX:  $\sigma_{FM} SAL, BUD (((ENO, Integer, 4), (Salary, Integer, 5)),$   
 $((DNO, Integer, 3), (Budget, Real, 6)))$

This statement defines the format of two CP's, SAL(salary) and BUD(Budget).

ii) Create:

This statement takes the form,

$$\sigma_{CR} \text{ cp } (<DATA>)$$

It creates the CP with the data specified; the data must follow the format specified earlier by a format statement.

iii) Destroy:

This statement is of the following syntax,

$$\sigma_{DS} \text{ } c_1, c_2, \dots, c_n$$

All the CP's specified would be physically deleted from the data base.

A detailed Backus-Naur form (BNF) syntax of WCRL is presented in the Appendix A. The syntax includes the recursion feature of retrieval part of WCRL.

In the next section, a higher level language for the Data Base Administrator is developed.

#### 4.4 Data Base Administrator Language (DBAL)

The concept of a Data Base Administrator (DBA) was first introduced in the recommendations of CODASYL [CODASYL 71] and ANSI [ANSI 75] committees. Since then, the functions, to be performed

by a DBA, have gone through a number of changes due to trends in the evolution of DBA concept. A detailed review on this, may be found in [DeBLASIS 78]. Some of the major functions performed by a DBA are the following.

- i) The Data base design and implementation,
- ii) Incorporating changes in the data base from time to time.
- iii) Monitoring the performance of the data base.

In order to support these functions, a high level language for DBA is developed in this section. So far, no dedicated language for DBA has been developed in the literature. Though WCRC supports DBAL at the external level, it allows the DBA to act directly on the conceptual level. The high level, english-like DBAL makes the task easier for the DBA. The translation of DBAL into WCRL is carried out at the external processor. The subsequent sections discuss the details of this language.

#### 4.4.1 DBAL - Data Definition (DDL)

The data definition part includes the facilities for defining the conceptual schema during the design process. The statements to define entity/relationship sets, attributes and value sets are listed below.

- i) DEFINE ATTRIBUTE <attribute name> ON  
VALUE\_SET <value set name>

- ii) DEFINE VALUE\_SET <value set name> OF  
 DATA TYPE <data type> WITH  
 PREDICATE (<allowable values, units>)
- iii) DEFINE ENTITY\_SET <entity set name> WITH  
 ATTRIBUTES (<list>)  
 KEY <key name>
- iv) DEFINE RELATIONSHIP\_SET <relationship set value> ON  
 ENTITIES (<list>) OF  
 TYPE (<association type>)  
 ATTRIBUTES <list>
- v) DEFINE WEAK\_ENTITY\_SET <entity set name> WITH  
 ATTRIBUTES <list>  
 KEY <key name> DEPENDENT ON  
 RELATIONSHIP relationship name

The predicate in value set definition allows the DBA to specify the integrity constraints such as allowable or permitted values. Further, any other integrity or security constraints may be specified by the following statements.

- vi) DEFINE ON <object name>  
 CONSTRAINT <expression>

where <expression> is of the form

<object name> <opr> <opd> and

<opr>: = set comparison or relational operator

<opd>: = <aggregate function> <object name> | <constant>

```

vii) GRANT   TO      (<user numbers>)
        ON      <object name>
        ACCESS (<access type list>)

```

This command may be used to specify the access privilege, such as read, write, etc., granted to the users.

An example to illustrate schema definition is shown in Fig. 4.7. The corresponding DDL statements are given below.

Example: Schema Definition

```

DEFINE  VALUE_SET  Name of
        DATA TYPE  String (10) WITH
        PREDICATE   Null ;

DEFINE  VALUE_SET  Amount OF
        DATA TYPE  Integer (4) WITH
        PREDICATE   (10K - 60K, Dollars) ;

DEFINE  VALUE_SET  E#  OF
        DATA TYPE  Integer (4) WITH
        PREDICATE   (6000 - 9999)

DEFINE  VALUE_SET  D#  OF
        DATA TYPE  Integer (4) WITH
        PREDICATE   (1 - 40) ;

DEFINE  VALUE_SET  Capital OF
        DATA TYPE  REAL (8)   WITH
        PREDICATE   (100K - 2M, Dollars) ;

```



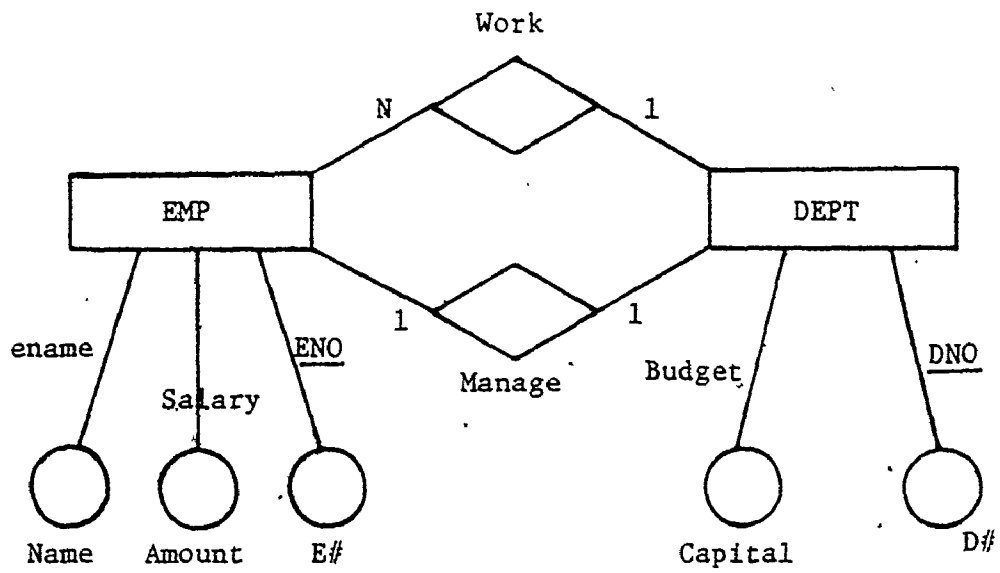


Fig. 4.7 Conceptual Schema of an Example Data Base

```

DEFINE ATTRIBUTE Ename ON
      VALUE_SET Name ;
DEFINE ATTRIBUTE Salary ON
      VALUE_SET Amount ;
DEFINE ATTRIBUTE ENO ON
      VALUE_SET E# ;
DEFINE ATTRIBUTE Budget ON
      VALUE_SET Capital ;
DEFINE ATTRIBUTE DNO ON
      VALUE_SET D# ;
DEFINE ENTITY_SET Emp WITH
      ATTRIBUTES ENO, ename, salary, Age
      KEY ENO ;
DEFINE ENTITY_SET Dept. WITH
      ATTRIBUTES DNO, Budget
      KEY DNO ;
DEFINE RELATIONSHIP_SET Work ON
      ENTITIES (Dept, Emp) OF
      TYPE (Binary, 1:N);
DEFINE RELATIONSHIP_SET Manage ON
      ENTITIES (Dept, Emp) OF
      TYPE (Binary, 1:1) ;

```

(Note: These relationships do not have any attributes)

Some Constraints:

```

DEFINE ON Emp. Salary
        CONSTRAINT (SAL 15K) ;

DEFINE ON Manage
        CONSTRAINT (EMP(Manage)_CEMP) ;

GRANT TO (U1, U2, U3)
      ON (EMP)
      ACCESS (Read, Modify) ;

```

Note, all these DDL statements have equivalent commands in WCRL-DDL. Therefore, they can easily be translated into WCRL commands.

4.4.2 DBAL - Data Manipulation (DML)

This facility allows the DBA to make changes to the schema as well as the data, due to data base growth. The schema change commands take the following form:

- i) ADD <object name> TO <object type>
- ii) DELETE <object name> FROM <object type>
- iii) SPLIT <object type> <object name> INTO <list>  
       QUALIFICATION <expression>
- iv) MERGE <object type> <list> INTO <object name>
- v) SHIFT (<mode>) <object name> TO <object name>
- vi) RENAME <object name> TO <object name>

where <object type> denotes entities/relationships/attributes/value sets,  
 <object name> is the corresponding name,

<list> is a set of two object names,

<expression> is of the form similar to the expression described in the previous section, and,

<mode> is either low or high

The 'add' command can be used to expand the schema, by adding an object to the existing conceptual schema, while 'delete' may be used to do the opposite. The 'split' operation may be employed to effect splitting of an object. For example, 'split' could cause the entity, employee to be divided into two entities, namely, managers and the managed. On the other hand, 'merge' can be used to combine two objects of the same type. The 'shift' operation is a unique operation, which changes a value set into an entity or an attribute into a relationship (high mode) and vice versa (low mode). Needless to say, the 'rename' operation allows for renaming of the objects in the schema. The use of these commands are further illustrated in Fig. 4.8a - 4.8d, borrowing the example from Fig. 4.7.

Example: To start with, consider a primitive data base shown in fig. 4.8a, consisting of only employees and departments. After some time, the DBA wants to expand the data base to include the information regarding the location of the departments. This can be accomplished by adding a new entity 'city' as shown in fig. 4.8b. Further, if it becomes necessary to distinguish between 'home phone' and 'office phone', the attribute phone may be split into two, as shown in fig. 4.8c. Furthermore, if the need arises to keep the information about the car owned by the employee, the shift operation may be utilized

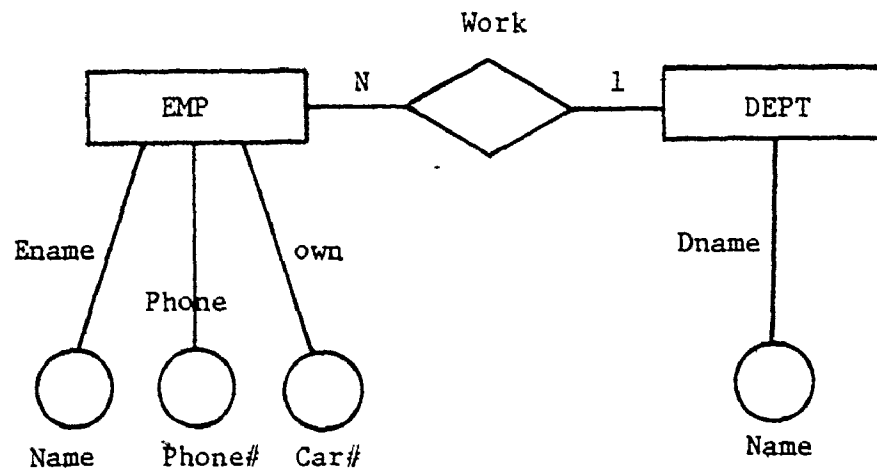
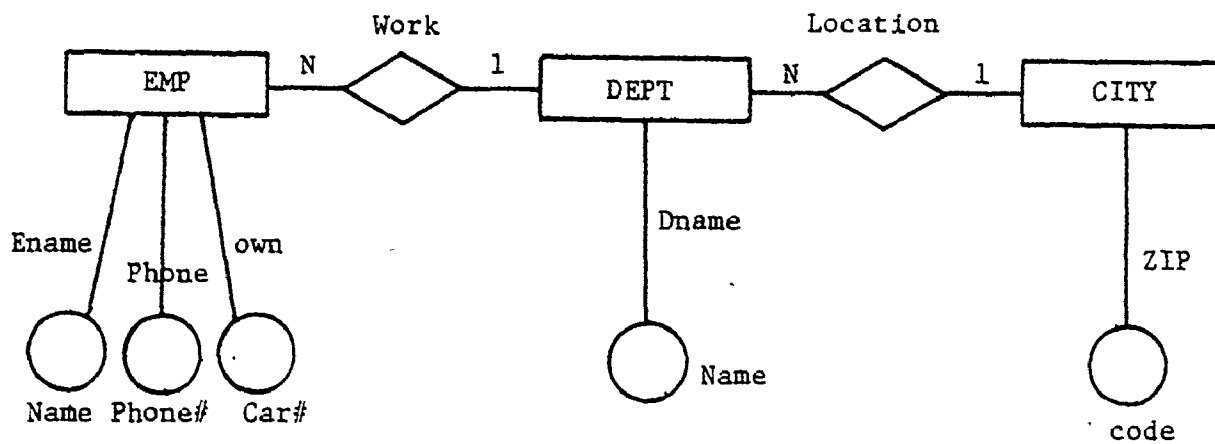
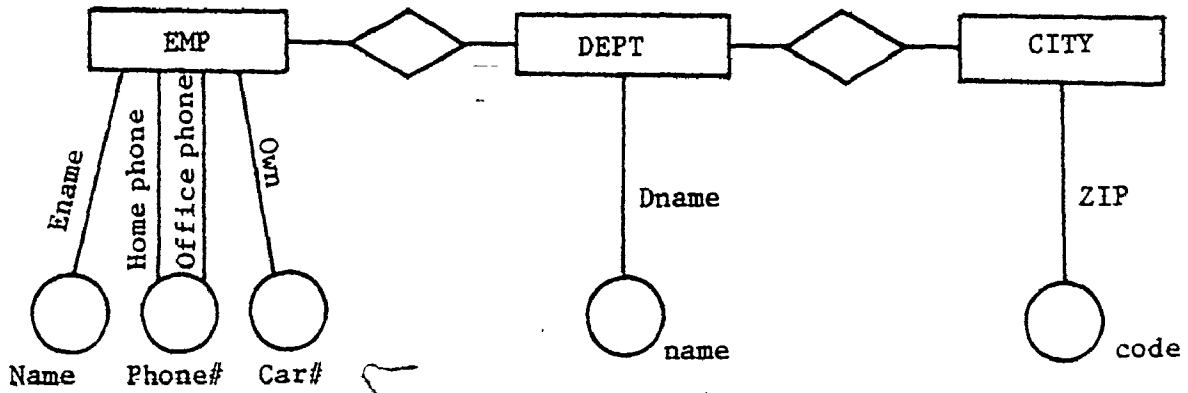


Fig. 4.8 a An Example Data Base



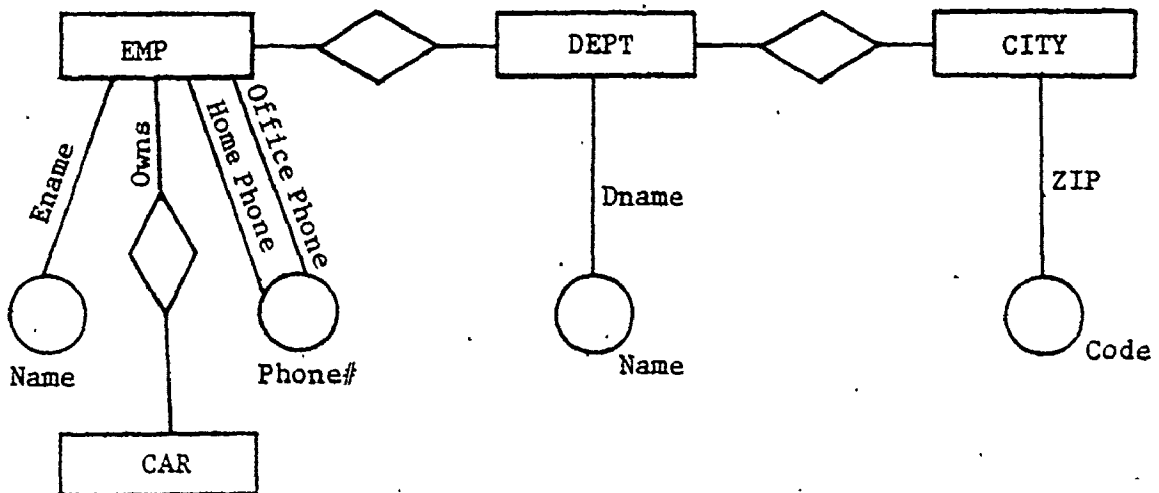
ADD City to Entities  
 ADD Location to Relationships

Fig. 4.8 b The Data Base After Adding Entity 'City'



SPLIT Attribute Phone INTO (Home Phone, Office Phone)

Fig. 4.8 c The Data Base After Splitting 'Phone'.



SHIFT (High) 'Own' To 'Owns'  
 SHIFT (high) 'Car#' To 'Car'

Fig. 4.8 d The Data

to promote the value set 'car' to an entity 'car', as shown in Fig. 4.8d.

Thus, schema changes can be handled by the DBA using commands i) through vi).

All these schema change commands, except 'add', can be expressed as  $\sigma_{cs}$  operation of WCRL-DDL. For example, consider the command, shift (High) 'own' To 'car'. The equivalent operation in WCRL-DDL is:

$$\begin{aligned} \sigma_{cs} \text{ own (attribute) } (\phi) \\ = e('car') \end{aligned}$$

Similarly, 'split', 'merge', 'delete', and 'rename' commands are expressible as special cases of  $\sigma_{cs}$ . Examples for 'split', 'merge' and 'delete' are already shown in section 4.3.2. The WCRL equivalent of 'rename' is as follows:

$$\sigma_{cs} \text{ 'Employee' (entity) } (\phi) = e \text{ ('worker')}$$

Depending upon the object type, the WCRL equivalent of 'add' command may take the form of any of statements i, ii, iii or iv of section 4.3.2.

For updating the data, the DBA is supplied with the following statements, which are self-explanatory.

- vii) INSERT INTO <object name> <tuple|block of tuples>
- viii) DELETE FROM <object name>(<tuple>|WHERE<condition>)
- ix) MODIFY <object name> (<tuple>|WHERE<condition>)  
BY <tuple>

Here, condition indicates which tuples should be selected. The syntax of these commands may be found in Appendix B. These three commands have equivalent DML statements in WCRL (section 4.3.3).

In order to ensure good performance, the DBA has to check the data base from time to time. For this purpose, the DBA is provided with a data retrieval command. The general form of the command is shown below.

```
RETRIEVE <argument list>
WHERE    (<conditions>)
```

The argument list specifies the attributes of an entity or a relationship to be retrieved. If attributes are not specified explicitly along with the entity (or relationship), all the attributes will be retrieved. While retrieving an entity/a relationship, the navigation may also be specified in the argument list by the USING clause, which consists of a set of entities and relationships, along a particular path in the E-R model of the data base. The navigation allows the DBA to retrieve the same data in several paths and helps to check the accuracy and consistency of the data base. The tuples, to be retrieved, are restricted by a condition. The condition consists of several expressions connected by boolean operators, where each expression in turn may involve set and aggregate functions, and relational and set comparison operators. A detailed syntax of the statement is presented in Appendix B. Some example queries are provided below to demonstrate the use of the command.



Example: Consider the data base shown in Fig. 4.7.

- i) List all employee names whose salary is greater than \$10,000.

```
RETRIEVE   All (EMP.(Ename))
WHERE      (EMP. Sal > 10,000)
```

- ii) Find average salary of the employees in the department where  
'D·NO = 50' and 'Budget ≤ 100,000'

```
RETRIEVE   AVG (EMP.(Sal)) USING Dept, Work, Emp
WHERE      (DEPT. DNO = '50') AND
              (DEPT. Budget ≤ 100,000)
```

- iii) List all employee names and their salary who work in the  
same department as 'SMITH'

```
RETRIEVE   DEPT. (DNO [X]) USING Emp, Work, Dept.
WHERE      (EMP. Ename = 'SMITH')
RETRIEVE   All (EMP. (Ename, Sal)) USING Dept, Work, Emp
WHERE      (DEPT. DNO = X)
```

#### 4.4.3 DBAL - Storage Definition (SDL)

The SDL commands provide the DBA with facilities to define the storage organization of data. They allow him/her to specify the internal level data structure as well as the mapping between conceptual level and the internal level. These commands are listed below.

- 1) DECLARE\_CP <canonical partition name>  
AS <first constituent, second constituent>

- ii) DECLARE\_ER <entity name>  
       AS        <CP name list> KEY <key name>
- iii) DECLARE\_RR <Relationship name>  
       AS        <CP name list> KEY <key name>

The first command defines the constituents, of a canonical partition, which are nothing but the attributes in the E-R model. They are already defined using DDL statements. The last two commands specify, the entity and relationship sets in terms of CP's. Note the E-R model guarantees loss less join of the CP's to form entity and relationship relations.

A detailed BNF syntax of DBAL can be found in Appendix B.

#### 4.5 Query Translation

As mentioned in the previous chapter, the queries in the user languages, namely, SEQUEL, LMS Data Sublanguage and LSL, are translated by the external processor into queries in WCRL. In addition to these languages, the external processor must also translate the DBAL commands into equivalent commands in WCRL. In this section, the translation of these languages is illustrated with an example.

Consider a slightly modified version of the conceptual schema corporate data base (Fig. 4.1) as shown in Fig. 4.9. For the sake of simplicity, it is assumed that, the users in all models view the whole data base instead of only parts of it. The external schemas corresponding to the conceptual schema, in relational, network and hierarchical models are shown in Figs. 4.10 - 4.12. The canonical partitions

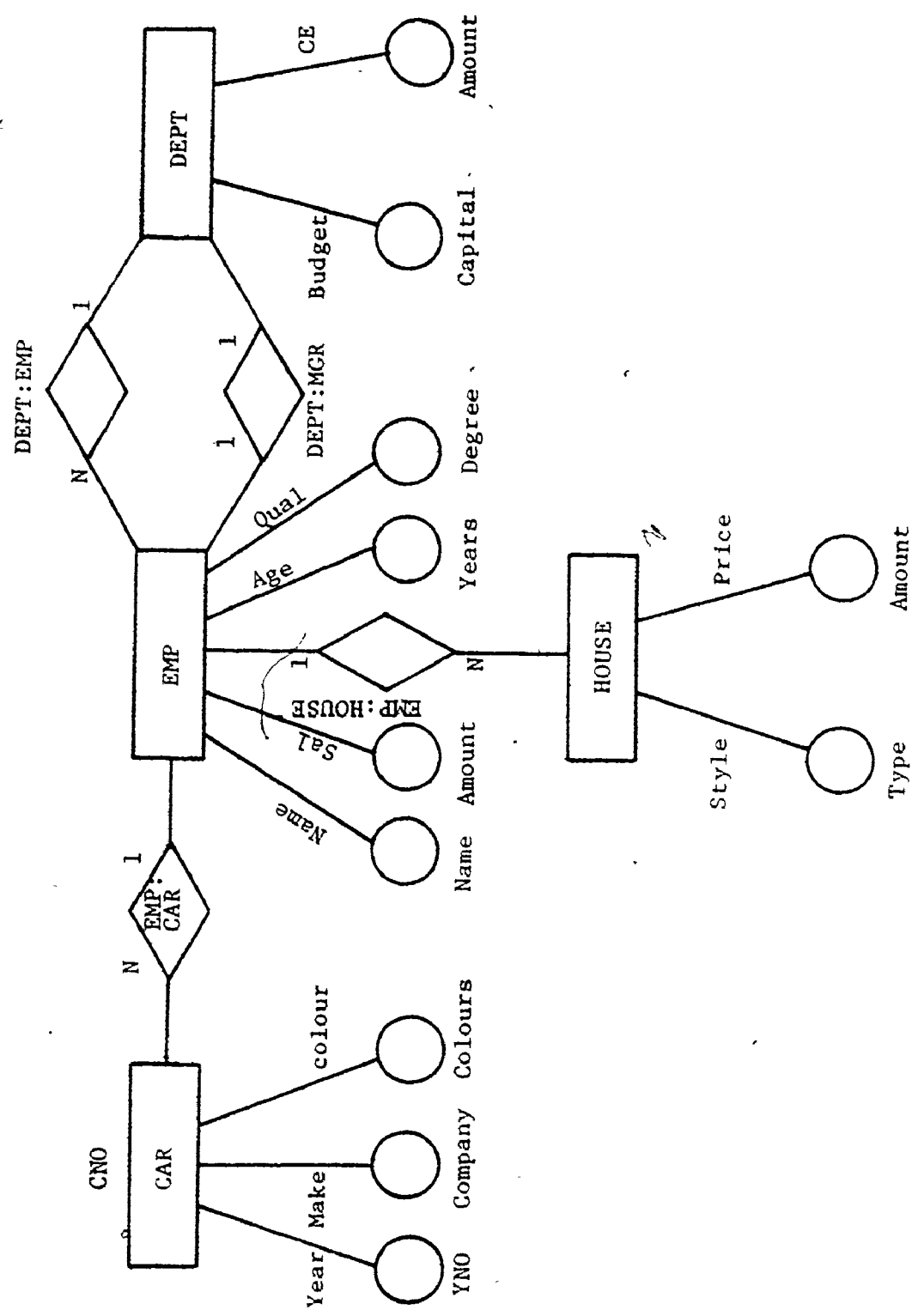


Fig. 4.9 E-R Diagram of a Corporate Data Base

DEPT (DNO, Budget, CE, Mgr)

EMP (ENO, Name, Sal, Age, Qual, DNO)

CAR (CNO, Year, Make, Colour, ENO)

HOUSE (HNO, Type, Price, ENO)

Fig. 4.10 The External Schema in Relational Model

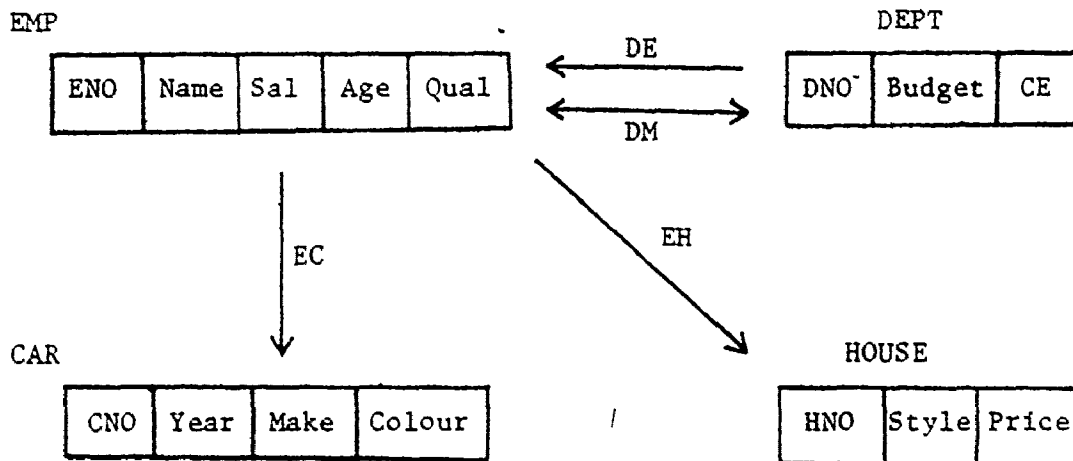


Fig. 4.11 The External Schema in Network Model

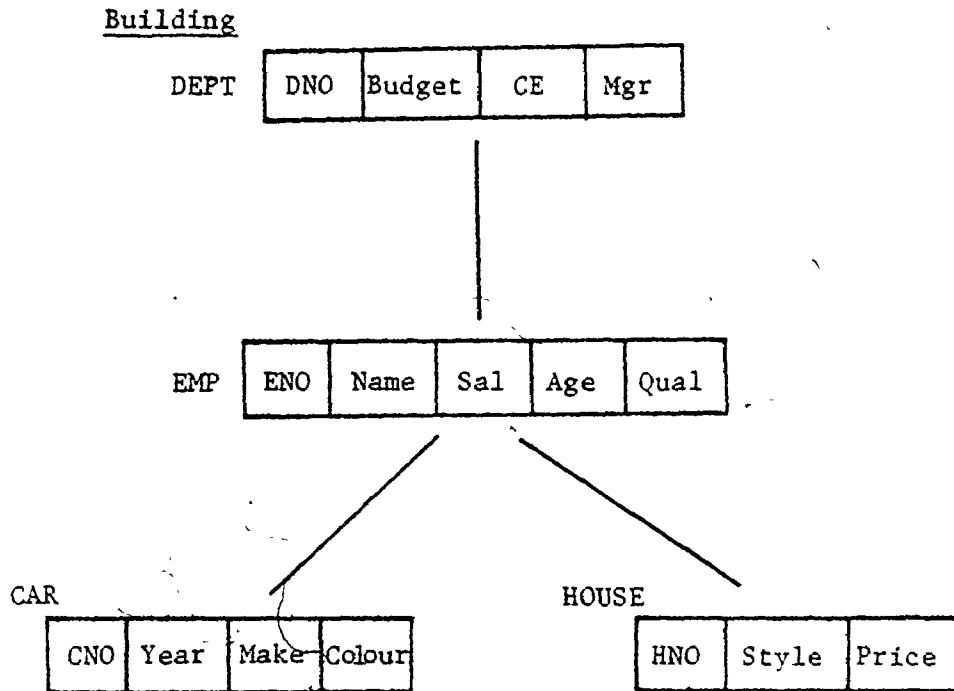


Fig. 4.12 The External Schema in Hierarchical Model

$w_1$ [ENO; Name] $w_2$ [ENO; Sal] $w_3$ [ENO; Age] $w_4$ [ENO; Qual] $w_5$ [DNO; Budget] $w_6$ [DNO; CE] $w_7$ [HNO; Style] $w_8$ [HNO; Price]	$w_9$ [CNO; Year] $w_{10}$ [CNO; Make] $w_{11}$ [CNO; Colour] $w_{12}$ [ENO; CNO] $w_{13}$ [DNO; ENG] $w_{14}$ [DNO; ENO] * (Mgr) $w_{15}$ [ENO; HNO] $w_{16}$ [DNO; BNO]
--	--

Fig. 4.13 The PCP's Corresponding to Fig. 4.9

(CP's) stored at the internal level, corresponding to the E-R model of the data, are shown in Fig. 4.13.

A typical query is expressed in these various languages and the corresponding translation into WCRL is as given below.

#### Example Query

Find the employees who live in houses that cost more than \$80,000 and who earn more than \$25,000, the brand of the cars owned by them, and the budget of the departments they work in. (i.e. report house price, employee name, car make and the department budget).

#### SEQUEL:

Based on the relational schema shown in Fig. 4.10, the user would write the query in SEQUEL as follows [CHAMBERLIN 76].

```

SELECT   HOUSE·price, EMP·name, CAR·make, DEPT·budget
FROM     HOUSE, EMP, CAR, DEPT
WHERE    (HOUSE·price > '80,000') AND
           (EMP·sal > '25,000') AND
           (HOUSE·ENO = EMP·ENO) AND
           (CAR·ENO) = EMP·ENO) AND
           (EMP·DNO) = DEPT·DNO)

```

#### LSL:

A user in network model would generate the same query, based on Fig. 4.11, as follows [TSICHRITSIS 76].

```

SELECT HOUSE
WHERE HOUSE.price > '80,000'
KEEP HOUSE. price
LINK WITH EH TO EMP
SELECT EMP
WHERE EMP.sal > '25,000'
KEEP EMP. name
LINK WITH EC TO CAR
KEEP CAR.make
LINK FROM EMP WITH DE TO DEPT
KEEP DEPT.budget

```

IMS Data Sublanguage:

Based on the hierarchical view of the data base shown in Fig. 4.12, another user would write the same query as follows

[DATE 76]

```

      GU   BUILDING
A     GN   BUILDING
B     GNP  DEPT * D
C     GNP  EMP (EMP.sal > '25,000')
      If not found GOTO C
D     GNP  HOUSE (HOUSE.price > '80,000')
      If not found GOTO D
      GNP  CAR

```

(Add DEPT. budget, EMP. name, HOUSE.price, CAR.make  
to Result list)

GOTO B

GOTO C

It may be noted that, the above program is not based on the genuine IMS DL/I [IBM 75] but on the hypothetical syntax given in [DATE 76], for the sake of simplicity.

DBAL:

If the DBA writes the same query in DBAL based on the E-R model shown in Fig. 4.9, the query would appear as given below.

RETRIEVE EMP. name, HOUSE. price USING Emp:House,  
CAR.make USING Emp:Car,  
DEPT.budget USING Emp:Dept.  
WHERE (EMP. sal > '25,000') AND  
(HOUSE.price > 80,000)

Finally, the query in all these languages, when translated into WCRL, would appear as follows.

WCRL:

$$\sigma_{sr} \{ \sigma_{cj} w_1 [ENO; NAME] w_2 [ENO; SAL] (w_1 \cdot ENO = w_2 \cdot ENO \wedge w_2 \cdot SAL > 25,000) \}$$

$$(ENO; NAME) = c_1 (ENO; NAME)$$

$$\sigma_{sr} \{ \sigma_{bj} w_{15} [ENO; HNO] w_8 [HNO; PRICE] (w_{15} \cdot HNO = w_8 \cdot HNO \wedge w_8 \cdot PRICE > 80,000) \}$$

$$(ENO; PRICE) = c_2 (ENO; PRICE)$$



$$\sigma_{sr} \{ \sigma_{cj} w_{13} [DNO; ENO] w_5 [DNO; BUDGET] (w_{13} \cdot DNO = w_5 \cdot DNO) \} (ENO; BUDGET) =$$

$$C_3 (ENO; BUDGET)$$

$$\sigma_{sr} \{ \sigma_{bj} w_{12} [ENO; CNO] w_{10} [CNO; MAKE] (w_{12} \cdot CNO = w_{10} \cdot CNO) \} (ENO; MAKE) =$$

$$C_4 (ENO; MAKE)$$

$$\sigma_{sr} \{ \sigma_{cj} c_1 [ENO; NAME] c_2 [ENO; PRICE] c_3 [ENO; BUDGET] c_4 [ENO; MAKE]$$

$$(c_1 \cdot ENO = c_2 \cdot ENO \wedge c_2 \cdot ENO = c_3 \cdot ENO \wedge c_3 \cdot ENO = c_4 \cdot ENO) \} (\phi; NAME,$$

$$PRICE, BUDGET, MAKE) = c_5 (\phi; NAME, PRICE, BUDGET, MAKE)$$

It may be noticed that, some of the WCRL statements are independent of one another while the others depend upon the sequence of statements before. Those statements which do not depend on others may be executed simultaneously on the CP's. The rest may be executed in the proper sequence. These WCRL statements will again be translated into machine executable hardware primitives, i.e. WCR machine language (WCRML) commands, before they are submitted for execution to the internal level. The WCRML commands and the translation of WCRL into WCRML are provided in Chapter V.

Note: Both WCRL and DBAL can handle partial relationships. However, this would lead to propagation of null values in the data base operations. Handling null values is still a grey area and it falls out of the scope of this thesis. Also, incorporation of null values would call for extension of LSL, SEQUEL and IMS data sublanguage. Some work in this area is available in [CODD 79] and [VASILLIOU 80].

#### 4.6 Query Analysis

In general, the higher level languages allow writing queries with complex conditional expressions [BOYCE 75], [CHAMBERLIN 74], [COPELAND 74]. Such queries would take a great deal of time if they are executed as they are. For this reason, the queries are first analysed and 'rephrased' before they are submitted for execution. Such improvements during query analysis are called query optimization. In this section, the conditional expressions involved in WCRL commands are examined and methods for optimizing some of the expressions are explored.

In WCRL, each command has an operation to be performed, a data specification section indicating which data is to be operated upon, and a qualification specifying the conditions that must be met before the operations can be fulfilled. Among these, the qualification forms the heart of a command. The qualifications in WCRL commands are primarily of two types, type I and type II:

- 1) The general format of the type I, is as follows

<attribute> <comparator> <operand>

where <attribute> is an attribute name in an EWCR or a CP,

> <comparator> is '=',

<operand> is one of the following:

- a) <aggr func> <attribute> ,
- b) <aggr func> <attribute> <aop> <aggr func> <attribute>
- c) constant

Here, <aggr func> refers to one of the following,

- i) SUM = sum of all appropriate attribute values,
  - ii) MIN = minimum of all appropriate values of the attribute,
  - iii) MAX = maximum of all appropriate values,
  - iv) CAR = number of distinct values of the attribute
  - v) AVG = average of all appropriate attribute values
- and the <aop> is one of (+, -, x, ÷).

It may be noticed that, the right-hand side of this type of qualification is always a constant, after it has been evaluated. Therefore, it takes only one interrogation or comparison to check the validity of the qualification.

- ii) The type II qualification or conditional expression in WCRL consists of a set of conditions connected by boolean operators ( $\wedge$ ,  $\vee$ ,  $\neg$ ) where each condition is of the type:
  - a) <attribute> <opr> <constant>
  - b) <set function> <attribute> <opr> <set function> <attribute>
  - c) <attribute> <copr> <attribute>

where <opr> is one of the relational operators (=,  $\neq$ , >, <,  $\geq$ ,  $\leq$ ),

<copr> is one of relational or set operators (=,  $\neq$ , >, <,  $\geq$ ,  $\leq$ ;

$\equiv$ ,  $\neq$ ,  $\supset$ ,  $\subset$ ,  $\supseteq$ ,  $\subseteq$ )

and set function is one of set functions (current, some, all)

EX: ( $A \geq 1000$ ), ( $B < C$ ), (current  $A >$  some  $B$ ) abbreviated as  $A > B$ ,

(All  $A \leq$  some  $B$ ) denoted by  $A \cdot \cdot \leq \cdot B$ .

The type (IIa) condition is similar to type I, except that the operator involved in type (IIa) may be one of comparison operators.

The type (IIb) involving relational operators and set functions may have 54 permutations as shown in figure 4.14. However, these permutations could be transformed into 3 basic classes. For the sake of simplicity, only '=', > and < operators are considered and the equivalent forms for type (IIb) conditions involving these operators are shown in figure 4.15. The rest may be expressed on similar grounds. The equivalent forms of these conditions may be obtained as follows:

Consider, the condition 'Some A > All B'. As can be seen, it is equivalent to 'max (A) > max (B)'. The evaluation of the latter would take only one comparison, provided the max (A) and max (B) values are previously known; while the former would have taken at least n comparisons and almost mn comparisons, where m and n are the number of values in attributes A and B respectively. Sometimes, the words 'set' and 'element' are used interchangeably with attribute and value in this section. From Fig. 4.15, the type (IIb) conditions may be classified into 3 classes, based on the number of comparisons, as

- i) Class 1, which requires a small constant number of comparisons (1, 2 or 3) independent of the cardinalities of the sets A and B (22 permutations)
- ii) Class 2, which require  $O(n)$  or  $O(m)$  comparisons (4 permutations)
- iii) Class 3, which needs  $O(mn)$  (i.e.  $O(n^2)$ ) comparisons (1 permutation)

It is clear from the above analysis, that most of type (IIb)

		B								
		current			some			all		
A	current	>, <, =,	>., <., =.,	>.., <.., =..,	>., <., =.,	>.., <.., =..,	>., <., =.,	>.., <.., =..,	>., <., =.,	>.., <.., =..,
	<u>&gt;</u> , <u>&lt;</u> , <u>=</u> ,	<u>&gt;</u> ., <u>&lt;</u> ., <u>=</u> .,	<u>&gt;</u> .., <u>&lt;</u> .., <u>=</u> ..,	<u>&gt;</u> ., <u>&lt;</u> ., <u>=</u> .,	<u>&gt;</u> .., <u>&lt;</u> .., <u>=</u> ..,	<u>&gt;</u> ., <u>&lt;</u> ., <u>=</u> .,	<u>&gt;</u> .., <u>&lt;</u> .., <u>=</u> ..,	<u>&gt;</u> ., <u>&lt;</u> ., <u>=</u> .,	<u>&gt;</u> .., <u>&lt;</u> .., <u>=</u> ..,	
	•>, •<, •=,	•>., •<., •=.,	•>.., •<.., •=..,	•>., •<., •=.,	•>.., •<.., •=..,	•>., •<., •=.,	•>.., •<.., •=..,	•>., •<., •=.,	•>.., •<.., •=..,	
some	•>, •<, •=,	•>., •<., •=.,	•>.., •<.., •=..,	•>., •<., •=.,	•>.., •<.., •=..,	•>., •<., •=.,	•>.., •<.., •=..,	•>., •<., •=.,	•>.., •<.., •=..,	
•>, •<, •=,	•>., •<., •=.,	•>.., •<.., •=..,	etc.	etc.	etc.	etc.	etc.	etc.	etc.	
all	•>, •<, •=,	•>., •<., •=.,	•>.., •<.., •=..,	•>., •<., •=.,	•>.., •<.., •=..,	•>., •<., •=.,	•>.., •<.., •=..,	•>., •<., •=.,	•>.., •<.., •=..,	
•>, •<, •=,	•>., •<., •=.,	•>.., •<.., •=..,	etc.	etc.	etc.	etc.	etc.	etc.	etc.	

Figure 4.14 Permutations of Type IIb Condition

Condition	Equivalent	# of comparisons for original condition	# of comparisons for the equivalent
Current A > Current B	$a_1 > b_1$	1	1
Current A = Current B	$a_1 = b_1$	1	1
Current A < Current B	$a_1 < b_1$	1	1
Current A > Some B	$a_1 > \min(B)$	1 - n*	1
Current A = Some B	$(a_1 = b_1) \vee \dots \vee (a_1 = b_n)$	n	n
Current A < Some B	$a_1 < \max(B)$	1 - n	1
Current A > All B	$a_1 > \max(B)$	n	1
Current A = All B	$(a_1 = \min(B) = \max(B))$	n	2
Current A < All B	$a_1 < \min(B)$	n	1
Some A > Current B	$b_1 < \max(A)$	1 - m	1
Some A = Current B	$(b_1 = a_1) \vee \dots \vee (b_1 = a_m)$	m	m
Some A < Current B	$b_1 > \min(A)$	1 - m	1

(\*Note: 1-n denotes that it takes at least 1 comparison and at most n comparisons)

Figure 4.15 The Equivalent Forms of Type IIb Conditions

Condition	Equivalent	# of comparisons for original condition	# of comparisons for the equivalent
Some A > Some B	$\max(A) > \min(B)$	$1-mn$	1
Some A = Some B	$(a_1=b_1) \vee \dots (a_1=b_n) \vee \dots$ $\dots \vee (a_n=b_1) \vee \dots \vee (a_n=b_m)$	$1-mn$	$mn$
Some A < Some B	$\min(A) < \max(B)$	$1-mn$	1
Some A > All B	$\max(A) > \max(B)$	$n-mn$	$m+1$
Some A = All B	$(a_1=\min(B)=\max(B)) \vee$ $\dots \vee (a_m=\min(B)=\max(B))$	$n-mn$	1
Some A < All B	$\min(A) < \min(B)$	$n-mn$	1
All A > Current B	$\min(A) > b_1$	$m$	1
All A = Current B	$(b_1=\min(A)=\max(A))$	$m$	2
All A < Current B	$b_1 > \max(A)$	$m$	1
All A > Some B	$\min(A) > \min(B)$	$m-mn$	1
All A = Some B	$(b_1=\min(A)=\max(A)) \vee$ $\dots \vee (b_n=\min(A)=\max(A))$	$m-mn$	$n+1$
All A < Some B	$\min(A) < \max(B)$	$m-mn$	1
All A > All B	$\min(A) > \max(B)$	$mn$	1
All A = All B	$\min(A) = \max(A) =$ $\min(B) = \max(B)$	$mn$	3
All A < All B	$\max(A) < \min(B)$	$mn$	1

Figure 4.15 The Equivalent Forms of Type IIb Conditions (continued)

conditions involve only a few comparisons. Therefore, these conditions, in the queries, should be replaced by their equivalent forms. It may be noted that, the use of equivalents would save considerable number of memory interrogations, when partially associative memories are used for storing the data. Also, the minimum and maximum values can be obtained in these memories in just one interrogation.

The type IIc conditions would involve either relational operators or set operators. The conditions with set operators are the following:

- i)  $A \supset B$  (set A contains set B)
- ii)  $A \subset B$
- iii)  $A \not\supset B$
- iv)  $A \not\subset B$
- v)  $A \equiv B$  (set A is equal to set B)
- vi)  $A \neq B$
- vii)  $A \supseteq B$
- viii)  $A \subseteq B$

Out of these, only i, iii and v need to be considered. The rest can be derived from the three conditions and the boolean operators.

EX: 'A  $\neq$  B' is equivalent to  $\neg(A \equiv B)$ . These three may further be simplified as follows. 'A  $\supset B$ ' is equivalent to 'All B = Some A and CAR(A) > CAR(B)'. 'A  $\not\supset B$ ' is same as 'Some B > All A or Some B < All A'. 'A  $\equiv B$ ' can be expressed as 'All A = Some B' and CAR(A) = CAR(B)'.

Note, these are similar to type (IIb) conditions. In fact, they belong



to class 2, class 1 and class 2, respectively. Also, for the conditions involving set operators, the number of interrogations required to evaluate the condition would be of the order of  $O(n^2)$ ; i.e. they belong to class 3. These results are summarized below.

<u>Type of condition</u>	<u># of comparisons</u>
Type I	1
Type (IIa)	1
Type (IIb) & (IIc)	
-class 1	1-3
-class 2	$O(n)$
-class 3	$O(n^2)$

Up to this point, the discussion has focussed on query translation into WCRL commands and query analysis based on the conditional expressions in WCRL statements. In the next section, the translation of user views into views on the conceptual model, the consistency checking of these views, and update strategy adopted at conceptual level are discussed.

#### 4.7 View Translation, Consistency and Update Strategy

As mentioned earlier, the user views may be in relational, network or hierarchical models. These views must be mapped into views based E-R model by the external processor.

Even though the view translation is not handled at the conceptual level, it is discussed in this chapter because the mapping algorithms can be better appreciated in the context of the conceptual data model. In the following sections, firstly, the mapping algorithms from various views into E-R views are described and secondly, the concepts of consistency of views and update strategy are developed.

#### 4.7.1 Mapping Relational Views into E-R Model

The views in the relational model consist of a set of relations, where each relation satisfies the following conditions. All the non-key attributes are fully functionally dependent on the key attributes and the non-key attributes are independent of each other; i.e., the relations are in 3NF. These conditions are necessary to maintain semantic uniformity between external subschemas in relational model and the main schema in E-R model. Before the mapping rules are presented formally, a set of definitions are presented to facilitate further discussion.

Definition 4.10: A relation whose key (primary) does not contain a key of another relation is called a primary relation (PR).

Definition 4.11: A relation, whose primary key is fully or partially formed by concatenation of primary keys or other relations, is called a secondary relation (SR). A secondary relation whose key is formed fully by concatenation of primary keys of other primary relations is

said to be of type 1 (SR1). The secondary relations with primary key formed fully by concatenations of primary keys of primary and secondary relations are referred to as type 2 (SR2). The secondary relation whose key contains partially some independent attribute(s) are called type 3 (SR3). The key attribute(s) in secondary relations (SR2 and SR3), which is either independent or key of some primary relation, is referred to KAI or KAP type, respectively.

Definition 4.12: The non primary key attributes of a relation may be classified into three types. Those non primary key attributes of a relation which do not participate in any other relation are called NKA1 type. The non primary key attributes of a relation which form a part of some primary relation are termed as NKA2 type. The rest are called NKA3 type, i.e., the non primary key attributes that are part of some secondary relation and not involved in any primary relation.

Based on this formalism, the transformation rules, from relational views to E-R views are described below.

#### Transformation Rules

- 1) For each primary relation, define a corresponding entity and identify it by the primary key. Define its NKA1 type attributes as the attributes of the entity and set up the domains as value sets.
- 2) For each secondary relation, SR1, define a relationship among the entities involved, identify it by the primary key, and identify the association as binary or k-ary depending upon the



Output: {E}, a set of entities where  $E_i \in \{E\}$  is of the form  
 $E_i(\text{ID}, (A_i))$ , ID being the identifier of the  
entity and (A), the set of attributes of the entity.

{R}, a set of relationships where  $R_i$  is of the form  
 $R_i(\text{ID}, t, (A))$ , t being the type of the relation-  
ship.

Procedure TRANS\_RE (S,F)

begin

For each PR  $\in$  S do

begin

Define an  $E_i$ ;

$E_i(\text{ID}) \leftarrow \text{key}(\text{PR});$

$\{E\} \leftarrow E_i;$

For each domain  $D_i \in$  PR do

begin

If  $D_i \in \text{NKA1}$  then  $E_i(A_i) \leftarrow D_i$

else

begin

If  $D_i \in \text{NKA2}$  then

begin

Define a  $R_i$

Identify  $t_i$ ;

$R_i(\text{ID}) \leftarrow (\text{key}, D_i);$

$R_i(A_i) \leftarrow \phi;$

If  $R_i \notin \{R\}$  then  $\{R_i\} \leftarrow R;$

```

                                else (skip)
                                end
                                else (skip)
                                end
                                end
                                end;
For each SR2  $\in$  S do
    begin
        Define a weak  $E_i$ ;
         $E_i$  (ID)  $\leftarrow$  key (SR2);
         $E \leftarrow E_i$ 
        For each  $D_i \in$  SR2 do
            begin
                If  $D_i \in$  NKA1 then  $E_i(A_i) \leftarrow D_i$ 
                else (skip)
            end
        end;
    end;
For each SR3  $\in$  S do
    begin
        Define KAI as  $E_i$ ;
         $E_i$  (ID)  $\leftarrow$  KAI;
        Define a relationship  $R_i$ 
         $R_i$  (ID)  $\leftarrow$  key (SR2);
        Identify the type,  $t_i$ 
        If  $R_i \notin \{R\}$  then  $\{R\} \leftarrow R_i$ 
    end

```

```

    else (skip)
    For each NKAL  $\leftarrow$  SR2 do Ri(Ai)  $\leftarrow$  Di
end;
For each Ri  $\in$  {R} do
    begin
        If Ri is binary then
            begin
                If Ei  $\leftrightarrow$  Ej then Ri(ti)  $\leftarrow$  1:1
                else If Ei  $\rightarrow$  Ej then Ri(ti)  $\leftarrow$  N:1
                else Ri(ti)  $\leftarrow$  M:N
            end
        else Ri(ti)  $\leftarrow$  M:N
    end;
end (procedure)

```

As an example, consider the relational view shown in Fig. 4.16. The EMP relation describes a set of employees, giving the name, salary, manager, department number, and the commission for each employee. The DEPT. relation gives the department number and the location. The SALES and NEW SALES relations describe the items sold by the department in the past and the present and their volume. The relation SUPPLY gives the information about the companies that supply items to the departments and their volume. The relation CLASS describes the type of each item. The keys of these relations are marked in Fig. 4.16 by underlining. According to this information given in Fig. 4.16, the above

EMP (Name, sal, Mgr, Dept#, Comm)  
DEPT (Dept#, Building, Floor)  
SALES (Dept#, Item, Cvol)  
SUPPLY (Company, Item, Dept#, Vol)  
CLASS (Item, Type)  
NEW SALES (Dept#, Item, Volume)

Figure 4.16 A Relational View of a Hypothetical Data Base



algorithm yields the E-R view shown in Fig. 4.17. The primary relations EMP, DEPT, and CLASS give rise to three entities with the same names. The key attribute, company, in secondary relation, SUPPLY, defines the entity, COMPANY. All the secondary relations give rise to corresponding relationships as shown in the Fig. 4.17. Note, if interrelational dependencies are not specified, the algorithm may give rise to a disconnected E-R diagram.

#### 4.7.2 Mapping Network Views Into E-R Model

A network view may consist of several record types and links as in the general network model [TSICHRITZIS 77]. All the links are binary and are of type 1:1, 1:N or N:M. Some record types may have recursive links defined on themselves. A recursive link refers to a link defined on a single record type. For example, a link 'Component' is a recursive link defined on a record type, 'PART', implying that a part is made up of other parts. The transformation rules for network views described above, are the following.

##### Transformation Rules

- 1) For each record type, define a corresponding entity and identify the data items (d) as the attributes of the entity.

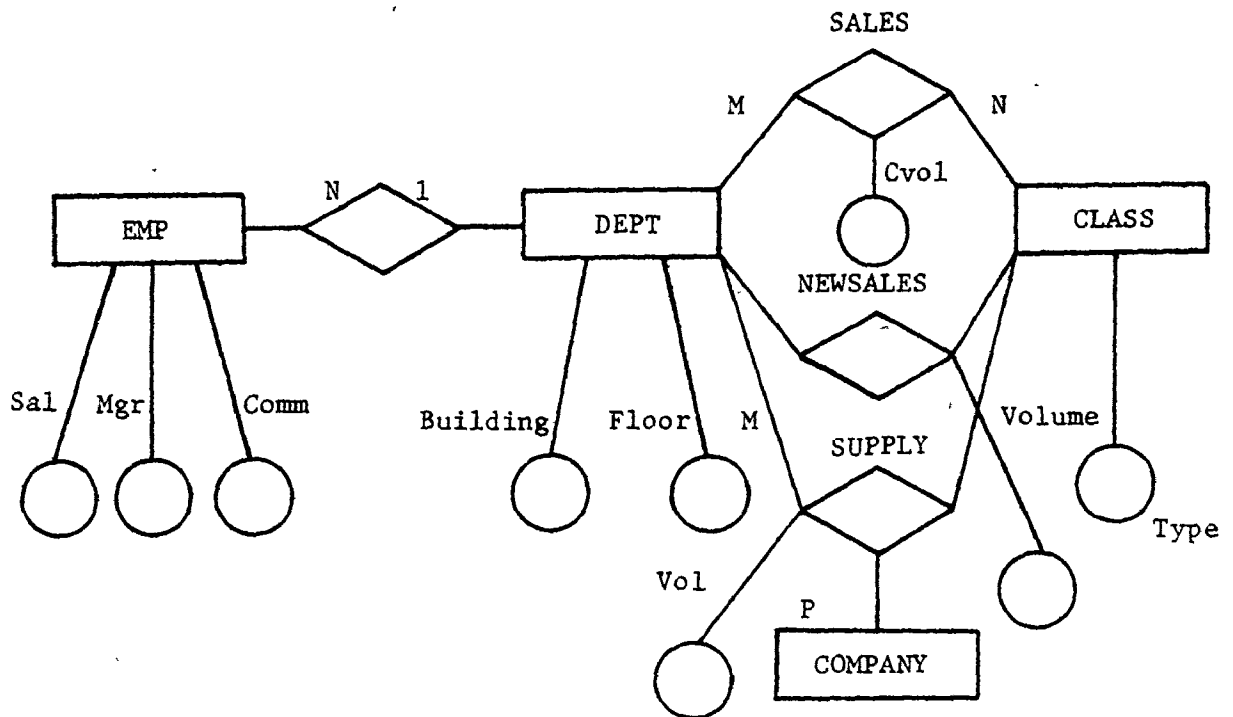


Fig. 4.17 The E-R View Corresponding to Fig. 4.16

- 2) For each link, define a relationship and label the association type (t).
- 3) For a recursive link, define a relationship on the same entity.

An algorithm for such a mapping is presented here.

#### The Algorithm

Input {RT}, a set of record types,

{L} , a set of links

Output {E} , a set of entities

{R} , a set of relationships

Procedure TRANS\_NE (RT,L)

Begin

For each  $\gamma \in RT$  do

begin

Define an  $E_i$

For each  $d_i \in \gamma$  do  $E_i(A_i) \leftarrow d_i$ ;

$E_i(ID) \leftarrow \text{key}(\gamma)$ ;

$\{E\} \leftarrow E_i$

end;

For each  $l \in L$  do

begin

Define a  $R_i$ ;

```

    Identify the type,  $t_i$ ;
     $R_i(\text{ID}) \leftarrow \text{key } (\gamma_i), \text{key } (\gamma_j);$ 
    ,  $\{\gamma_i \text{ and } \gamma_j \text{ are the record types linked by } \ell\}$ 
     $\{R\} \leftarrow R_i$ 
    end;
end (procedure)

```

As an example, consider a network view of some hypothetical data base as shown in Fig. 4.18. It consists of record types, EMP, PROJ, DEPT, SUPPLY, PART, describing the projects and the associated information and record types SPOUSE and CHILD describing personal information about the employees. The links between the records are labelled in Fig. 4.18. The corresponding E-R model obtained by applying the above algorithm is shown in Fig. 4.19. For simplicity, the attributes are not shown in the figure. Only entities and the relationships are identified.

#### 4.7.3 Mapping Hierarchical Views Into E-R Model

As mentioned earlier, the hierarchical views are represented by a set of trees, where each tree has a set of record types as nodes. The links from a parent node to its child nodes are information carrying. They are either 1:1 or 1:N type. It may be noted that, each tree in the hierarchical model is nothing but an acyclic, connected graph. [KNUTH 73]. The transformation rules for such a hierarchical forest into a E-R diagram are given below.

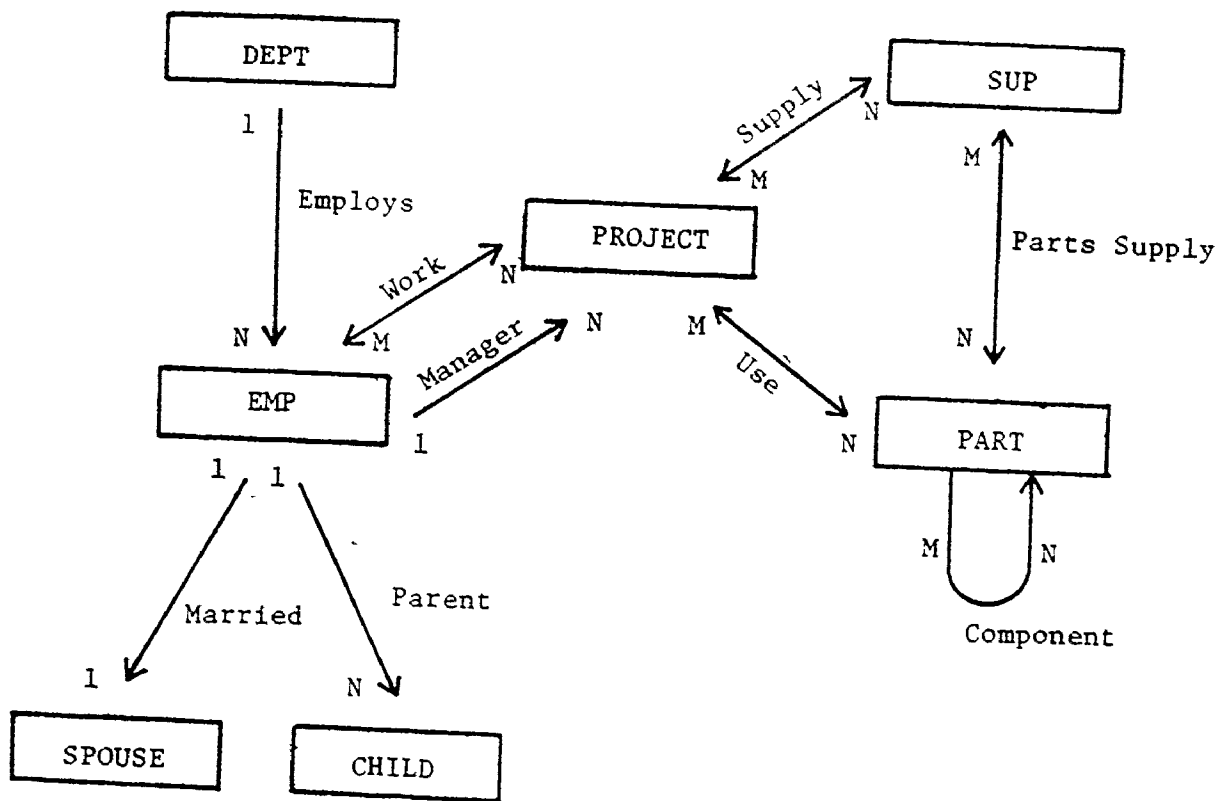


Fig. 4.18 A Network View of a Hypothetical Data Base

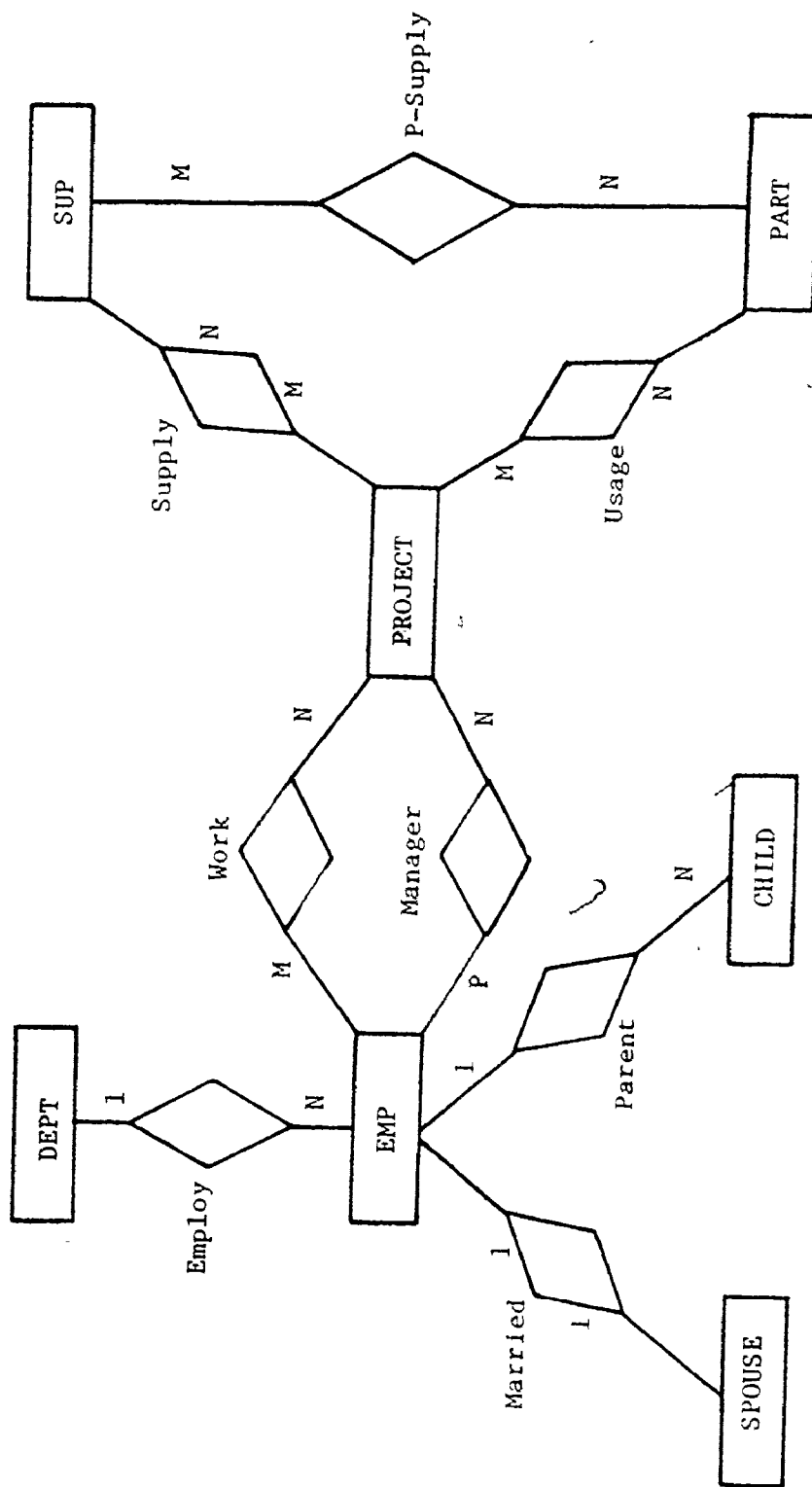


Fig. 4.19 The E-R Model of the Network View in Fig. 4.18

### Transformation Rules

- 1) If two trees in the forest have only common node, connect them at that node. Repeat this until the forest becomes a connected but not necessarily an acyclic graph.
- 2) Replace all nodes in the graph, by entities and edges by relationships. Identify the type of the relationships (either 1:N or 1:1).

An algorithm for the above mapping is given below. For the sake of simplicity, the details like attributes and relationships types are omitted.

### The Algorithm

Input {F}, A forest of hierarchical trees with edges ( $e_i$ )  
as 1:1 or 1:N and nodes  $n_i$

Output {E}, A set of entities,  
{R}, A set of relationships

Procedure TRANS\_HE (F)

begin

G  $\leftarrow$  t; (G is a graph; t is some tree in F)

For each tree  $s \in (F-G)$  do

begin

For each node  $n \in s$  do

begin

For each node  $m \in G$  do

begin

If  $n = m$  then NJOIN (G,s,n)

```

    end
  end
end;
For each node  $n \in G$  do
  REPLACE (n,E);
For each edge  $e \in G$  do
  REPLACE (e,R);
end (procedure)

```

NJOIN ( $G,s,n$ ) - is a function which performs the joining of the graph  $G$  and the tree  $s$  and the node  $n$ ,

REPLACE ( $n/e, E/R$ ) - is a function which replaces nodes by entities and edges by relationships and outputs the entities and relationships.

The above algorithm is illustrated by the example shown in Fig. 4.20. The hierarchical model in this figure consists of records STUDENT, COURSE, TEACHER, CLASS, ROOM. It is a view of some data base concerning a university. Each student in this view, may take several courses. Each course is taught by one teacher and held in one class room. But a teacher can offer more than one course and take several classes, where each class may have a number of students. Also, a class is taught by several teachers. These pieces of information are represented as hierarchies, as shown in fig. 4.20. The corresponding graph and the E-R diagram are shown in fig. 4.21 a and b. For the sake of simplicity, the attributes are not shown in any of these



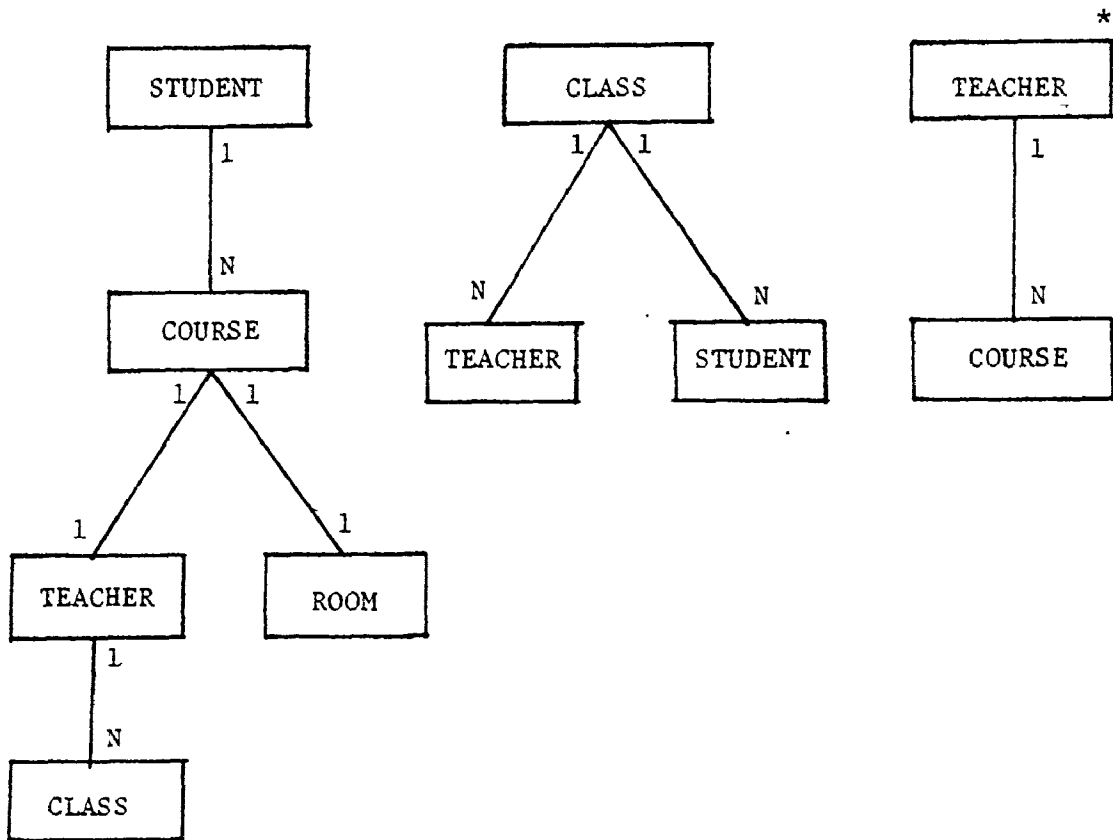


Fig. 4.20 A Hierarchical View of a University Data Base

(\* Some teachers may offer courses, but there may not be any classes taking them. Such information is shown in this tree)

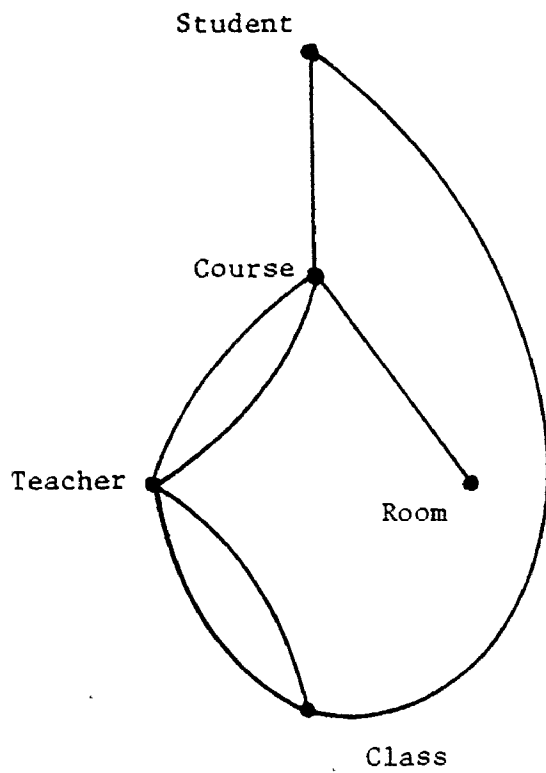


Fig. 4.21 a A Graph Corresponding to Fig. 4.20

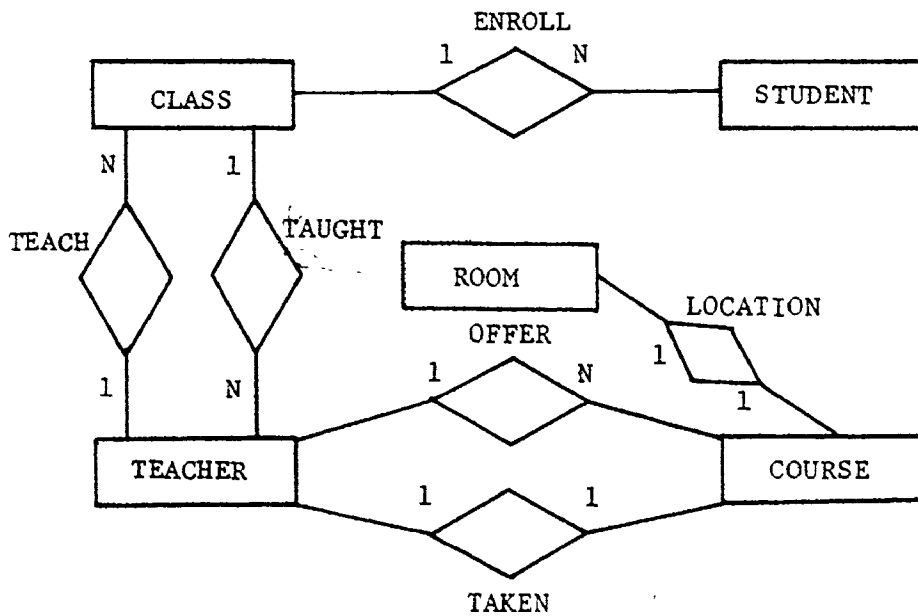


Fig. 4.21 b The E-R Model Corresponding to Fig. 4.20

diagrams. Note, the algorithm may sometimes give rise to disconnected E-R diagram.

In the following section, the consistency of E-R views, obtained by the above algorithms is discussed.

#### 4.7.4 View Consistency and Updates

In the previous section, it is shown how user views can be mapped into the conceptual schema. Now, an operation is permissible on such a view, only if it can be successfully mapped on to an operation(s) on the conceptual/main schema. In other words, given a main schema  $s_M$  and an instance,  $I$  of the data base, what subschemas  $s_V$  are permissible. Intuitively, a permissible view may be said to be consistent with the main schema. A similar problem is addressed elsewhere in a different context. For example, in [DALE 76] and [DALE 77], the main schema - subschema interaction is considered, in the context of hierarchical model. Also, in [ARORA 80A], a class of views are identified as consistent views, based on the functional dependencies in the underlying data base, in the context of the relational model. Here, we examine the consistency problem in the context of the conceptual level, where both the schema and subschemas are in the E-R model. The required formalism is developed below.

Definition - Consistency of View Mapping: A view mapping ( $f_V$ ) is said to be consistent with the main schema  $S_M (S, I, C)$ , where  $S$  is the schema,  $I$  is the instance and  $C$  is the set of constraints,

if no constraints in  $C$  are violated by the mapping,  $f_v$ .

Definition: A view is said to be retrievably consistent, if every query definable on view,  $s_v$ , is executable on the instance  $I$  of  $S$ . Such a view is called 'query equivalent' to the main schema.

The following views are query equivalent to the main schema:

- a) Subset views
- b) Derivable views

Definition: A view,  $s_v$  is called a subset view of a main schema,  $S_M$ , if, i) the entities in  $s_v$  are a subset of entities in  $S_M$ , ii) the relationships in  $s_v$  are a subset of relationships in  $S_M$ , and iii) for every entity and relationship in  $s_v$ , the set of attributes is a subset of the attributes of the corresponding entity or relationship in  $S_M$ .

Definition: A view,  $s_v$ , is called a derivable view of a main schema  $S_M$  if the entities and relationships in  $s_v$  are derivable by the following five operations from those in  $S_M$ .

- 1) Joining the relationships in  $S_M$  to obtain new relationships  
(R JOIN).
- 2) Joining entities through relationships to obtain new entities  
(E JOIN).
- 3) Shifting value sets into entities and attributes into relationships, and vice versa (SHIFT).

- 4) Breaking a k-ary relationship into a set of binary relationships (BREAK) and performing RJOIN on them.
- 5) Reconstituting the attributes of an entity or relationship, i.e., selecting only certain attributes (RECONSTITUTE).

As an example, consider the E-R model of a data base shown in Fig. 4.19. By joining (R JOIN) the relationships 'Employ' and 'Work' between entities DEPT, EMP and PROJECT, a new relationship may be obtained between DEPT and PROJECT. Similarly, a new entity may be obtained by joining (EJOIN) entities EMP and SPOUSE over the relationship 'Married'. The other operations SHIFT, BREAK and RECONSTITUTE are self explanatory.

The operations RJOIN and EJOIN are based on the 'natural' join. It is easy to see that, the queries defined on subset views are executable directly on the main schema. In the case of derivable views, the queries must be subjected to the derivation function ( $f_d$ ), before they are executable on the main schema. Therefore, to check the retrieval consistency of user views, it has to be decided whether the corresponding E-R views can be categorized as subset or derivable views of the main schema. If they do not belong to the above two types, they would be rejected as inconsistent views. Further, in order to derive the relationships in a view  $s_v$  from the relationships in main schema  $S_M$ , it is necessary to use 'BREAK' and 'RJOIN' operations. For this purpose, the behaviour of binary relationship relations under natural join is examined.

Figure 4.22 a and b shows the type of the relationship relation obtained by joining two relationship relations. Note, Fig. 4.22 b shows how partial and total relationship relations behave under the natural join operation. This information is useful in deciding whether the relationships in  $s_v$  agree semantically with the synthesized relationships.

Definition: A view  $s_v$  is said to be update consistent with schema  $S_M$ , if every update  $u$  on  $s_v$  is translatable into a set of updates  $\{v_i\}$  on  $S_M$ .

Definition: An update  $u$  on  $s_v$  is translatable onto  $S_M$ , if it does not violate any constraints  $C$  and the consequences of such an update are well defined.

The consequences are well defined for the following types of updates on the instance  $I_v$  of a view  $s_v$ .

- 1) Insertion of a tuple into entity/relationship in  $I_v$  causes insertion of these values into corresponding attributes of entity/relationship in  $I$  with null values in extraneous attributes. The extraneous attributes are those which occur in an object of  $I$  and not in the corresponding object of  $I_v$ , the object being an entity or a relationship.
- 2) Insertion of a tuple into weak entity causes a corresponding entry into the key of the supporting relationship (the other attributes get null values).

(a)

Type \ Type	1:1	1:N	M:N
1:1	1:1	1:N	M:N
1:N	1:N	1:N	M:N
M:N	M:N	M:N	M:N

(b)

Type \ Type	TOTAL	PARTIAL
TOTAL	TOTAL	PARTIAL
PARTIAL	PARTIAL	PARTIAL

Fig. 4.22 Behaviour of Binary Relationship Relations Under Natural Join

- 3) Deletion of a tuple from an entity in  $I_v$  causes deletion of specified values from the corresponding attributes in  $I$ . If the entity in  $S_M$  has no extraneous attributes, then the corresponding key value is also deleted. A null value is introduced into any relationship which involves this entity.
- 4) Deletion of a tuple from a relationship in  $I_v$  causes deletion of the values from the corresponding attributes. It also deletes the key of the relationship if there are no extraneous attributes in the corresponding relationship in  $S_M$ .
- 5) For modification, if the primary key (K) is modified, then the tuple is deleted and a new tuple is inserted. If a nonprimary key is modified, say, attribute A, then the modification is allowed only if the dependency  $K \rightarrow A$  is not violated; otherwise it is rejected.

We allow null values in the binary tuples. However, in query evaluation of a tuple, instead of three valued logic - 'True', 'False' and 'May be', we have only two valued logic - 'True' and 'False' with 'May be' treated as False. Each entity relation or relationship relation has an all null tuple with a distinct key value. This has the effect that no nulls can occur in any key domain and partial relationships can be treated as total. In view of this two valued logic, the following strategy is adopted for the basic relational algebra operations, join, projection, selection, union and difference. For join, the nulls on the joining columns are



dropped but nulls on the nonjoining columns are reported to the user. In case of projection, if the resultant relation has null values in all the columns, it is ignored. For selection, the null values do not qualify for any condition clause. For union and set difference, the null values are treated as any other value.

This strategy has the following advantages:

- i) Updates through user views can be allowed.
- ii) No modification of external languages - LSL, SEQUEL and IMS language - are needed.
- iii) Uncontrolled propagation of nulls in the system is avoided.
- iv) No special hardware for nulls is required.

In the above discussion, the entities and relationships are assumed to be organized as relations. In fact, these are organized as canonical partitions. So, it remains to show that any operation conceivable on these assumed relations can always be translated into a set of operations on the CP's. (It may be recalled that, the information regarding an entity/relationship relation is obtained by taking a natural join over the corresponding CP's). Now, the required result may be shown with the help of the following propositions. The notation here is similar to that of relational algebra [ULLMAN 80].

Proposition 4.1: Let  $R[KAB]$  be an entity/relationship relation, where K is the key and A, B are the attributes.

Let  $R(KAB) = R[KA] * R[KB]$

Let  $F$  be a selection clause such that  $F = F_1 \vee F_2$  where  $F_1$  is a selection clause on  $A$  and  $F_2$  on  $B$ . (For the sake of simplicity, no selection clause is assumed on key. However, it can always be incorporated on similar grounds). Then,

$$\sigma_F (R(\underline{KAB})) = (\sigma_{F_1} (R[\underline{KA}]) * R[\underline{KB}]) \cup$$

$$(\sigma_{F_2} (R[\underline{KB}]) * \sigma_{F_1} (R[\underline{KA}]))$$

Proof: Consider  $F = (A \theta a_1) \vee (B \theta b_1)$

where  $a_1 \in \text{Dom}(A)$  and  $b_1 \in \text{Dom}(B)$ .  $\theta$  could be one of the comparison operators ( $=, \neq, >, <, \geq, \leq$ ). Let  $\theta$  be in particular '='.

The proof would follow on similar lines for other operators. We shall prove that any tuple that satisfies LHS (left hand side) also satisfies the RHS and vice versa.

There are three types of tuples that satisfy the LHS:

a)  $\langle k_i a_1 b_l \rangle$ , b)  $\langle k_i a_j c_l \rangle$ , c)  $\langle k_i a_1 b_l \rangle$

Now, consider  $\langle k_i a_1 b_l \rangle \in R(\underline{KAB})$

$$\Rightarrow \langle k_i a_1 \rangle \in R[\underline{KA}]$$

$$\text{and } \langle k_i b_l \rangle \in R[\underline{KB}]$$

$$\langle k_i a_1 \rangle \text{ satisfies } \sigma_{F_1}$$

$$\langle k_i a_1 \rangle * \langle k_i b_l \rangle = \langle k_i a_1 b_l \rangle \in \text{RHS}$$

Similarly,  $\langle k_i a_j c_1 \rangle$  and  $\langle k_i a_1 b_1 \rangle$  also belong to RHS.

Any tuple that satisfies LHS also satisfies RHS

i.e.  $LHS \subseteq RHS$

To satisfy  $\sigma_F$ , a tuple  $\epsilon R[KA]$  should be of the type  $\langle k_i a_1 \rangle$ . Let some  $\langle k_i b_1 \rangle \epsilon R[KB]$ . This implies,  $\langle k_i a_1 b_1 \rangle \epsilon RHS$ . This tuple also satisfies LHS. Similarly, a tuple that satisfies  $\sigma_{F_1}$  belonging to  $R[KA]$  must be of the form  $\langle k_i a_1 \rangle$ . This implies that  $\langle k_i a_1 b_1 \rangle \epsilon RHS$ , if there is some tuple  $\langle k_i a_1 \rangle \epsilon R[KA]$ . Note,  $\langle k_i a_1 b_1 \rangle$  also satisfies  $\sigma_F$ , the LHS

$RHS \subseteq LHS$

Hence, the result.

Note: This result may be extended for a n-ary relation with a clause  $F = F_1 \vee F_2 \dots \vee F_n$ , on similar grounds. In this proposition, only relational operators are considered for  $\theta$ . However, the result can be extended for set comparisons operators.

Proposition 4.2: Let  $F$  be a selection clause on  $R(KAB)$  where  $R(KAB) = R[KA] * R[KB]$  and  $F = F_1 \wedge F_2$  such that  $F_1$  is a selection clause over  $A$  and  $F_2$  is selection clause over  $B$ . Then

$$\sigma_F(R(KAB)) = \sigma_{F_1}(R[KA]) * \sigma_{F_2}(R[KB])$$

Proof:  $\sigma_F(R) = \sigma_{F_1 \wedge F_2}(R)$

$$\begin{aligned}
&= \sigma_{F_1} (\sigma_{F_2}(R)) \quad (\text{cascade of selections}) \\
&= \sigma_{F_1} (\sigma_{F_2}(R[KA] * R[KB])) \\
&= \sigma_{F_1} (R[KA] * \sigma_{F_2}(R[KB])) \\
&= \sigma_{F_1} (R[KA]) * \sigma_{F_2} (R[KB])
\end{aligned}$$

Proposition 4.3: It is always possible to translate any query expressible in relational algebra based on entity/relationship relations into a set of queries on canonical partitions.

Proof: Let  $E_1(\underline{ABCD})$  and  $E_2(\underline{PQR})$  be two entity relations (or relationship relations). For the sake of simplicity, we assume that there are only two of these relations. Note  $E_1$  and  $E_2$  are similar to relations in relational model. Therefore, relational algebra could be applied to these relations. The basic operations in relational algebra are projection, selection, natural join, cartesian product, union and set difference. The other operations like  $\theta$ -join, division and intersection are expressible in terms of these operations [ULLMAN 80].

Let  $C_1(AB)$ ,  $C_2(AC)$ ,  $C_3(AD)$ ,  $C_4(PQ)$ , and  $C_5(PR)$  be the canonical partitions corresponding the  $E_1$ ,  $E_2$  and  $R$ . We show that the five relational algebra operations can be expressed as a combination of these operations on CP's. This would prove the proposition.

Projection (Case 1): When projection involves the key of a entity/relationship relation, it can be expressed as a natural join.

$$\text{EX: } E_1[(\underline{ABC})] = C_1(\underline{AB}) * C_2(\underline{AC})$$

Projection (Case 2): When the projection does not involve key, as in  $E_1[BC]$ , it can be expressed as follows:

$$E_1[BC] = (C_1(\underline{AB}) * C_2(\underline{AC})) [BC]$$

Selection (Case 1): When selection function is of the form  $F = F_1 \vee F_2$  where  $F_1$  is a selection clause, say, on  $Q$  and  $F_2$  is a selection clause on  $R$ .

$$\begin{aligned} \sigma_F (E_2 (PQR)) &= (\sigma_{F_1} (C_4(PQ)) * C_5 (PR)) \cup \\ &\quad (\sigma_{F_2} (C_4(PR)) * \sigma_{F_1} (C_5(PQ))) \end{aligned}$$

Selection (Case 2): When  $F = F_1 \wedge F_2$

$$\sigma_F (E_2 (PQR)) = \sigma_{F_1} (C_4(PQ)) * \sigma_{F_2} (C_5(PR))$$

The validity of these identities follows from propositions 4.1 and 4.2.

Natural Join: It is easy to see that

$$\begin{aligned} E_1 (ABCD) * E_2 (PQR) &= (C_1(AB) * C_2(AC) * C_3(AD)) * (C_4(PQ) * C_5(PR)) \\ &= C_1(AB) * C_2(AC) * C_3(AD) * C_4(PQ) * C_5(PR) \end{aligned}$$

Cartesian Product: Consider the cartesian product of  $E_1(ABCD)$  and  $E_2(PQR)$ . The equivalent expression is given by:

$$E_1(ABCD) \times E_2(PQR) = C_1(AB) * C_2(AC) * (C_3(AD) \times C_4(PQ)) * C_5(PR)$$

Union: Suppose the domains of A, B, C are compatible with those of P, Q, R. Consider union  $E_1[ABC]$  and  $E_2(PQR)$ .

$$\begin{aligned} E_1[ABC] \cup E_2(PQR) &= (C_1(AB) * C_2(AC)) \cup (C_4(PQ) * C_5(PR)) \\ &= (C_1(AB) \cup C_4(PQ)) * (C_2(AC) \cup C_5(PR)) \end{aligned}$$

This identity holds good because both A and P are keys of the CP's,  $C_1$  and  $C_2$ ,  $C_4$  and  $C_5$ , respectively.

Difference: Similar to the case of union, it can be easily shown that

$$\begin{aligned} E_1[ABC] - E_2(PQR) \\ = (C_1(AB) - C_4(PQ)) * (C_2(AC) - C_5(PR)) \end{aligned}$$

Thus, all the five relational operations on the entity/relationship relations can be expressed as a set of operations on the canonical partitions.

Proposition 4.4: Any query expressible in relational algebra on entity and relationship relations can be expressed in terms of WCRL commands on the canonical partitions.

Proof: The proof of this proposition follows from proposition 4.3 and relational completeness of WCRL. The relational completeness of WCRL is proved elsewhere [ARORA 80].

#### 4.8 Summary

The purpose of this chapter has been two-fold. First, a bottom-up design of the conceptual level based on the data model and the data language are described. Second, a theoretical basis for query analysis, query translation and view consistency are reported. The required algorithms for schema conversion and view translation are presented. These aspects would form the basis for implementing a conceptual processor, as described in Chapter III. The detailed software design of the conceptual processor is out of the scope of this thesis. In the next chapter, the details of the internal level will be discussed.

## CHAPTER V

### THE INTERNAL LEVEL

#### 5.1 Introduction

This chapter presents the details of storage structures, WCR machine language instructions and hardware organization of the internal level. For the overall configuration of the internal level the reader is referred to Fig. 3.1 in Chapter III.

The following design decisions have been taken regarding the hardware architecture of the internal level.

i) The storage organization is based on partially associative memories or logic-per-cell devices. These devices have been argued elsewhere, as cost effective alternatives to associative memories [SU 79A]. They reduce the system cost due to reduced software and increased memory utilization.

ii) An array of identical cells and query processors are used so that the cost of processing power is reduced, modular growth is made possible and processing efficiency is gained due to parallel processing.

iii) CCD memories may be used as the cell memories because they provide cost effective utilization of memory space, avoid problems associated with synchronization and mechanical driving, as in the case of magnetic disks, and provide less expensive packaging and



interconnections.

iv) The data is stored as binary canonical partitions rather than n-ary relations. These partitions provide greatest amount of flexibility in organizing the data to suit different data model needs at the external level. This approach is radically different from the other approaches in the literature.

The following section describes two possible implementations of canonical partitions.

## 5.2 The Data Structures

Since, in a canonical partition the WCR's may be of arbitrarily large size (number of tuples), it cannot be implemented directly as a data structure. Instead, two types of 'physical' data structures are proposed, in the section, for the storage of data on cell tracks. These are: a) pseudo canonical partition (PCP) - option I, b) pseudo canonical partition - option II.

### 5.2.1 PCP - Option I

Similar to a canonical partition, a PCP may be defined as follows: A partition of a binary relation  $R[A,B]$  is a pseudo canonical partition - option I, if

$$R[A,B] = \sum_{i=1}^n w_i[A_i;B_i]$$

where i)  $w_i[A_i;B_i]$  is a WCR for  $1 \leq i \leq n$

ii)  $A_i$  is a set with a single element for  $1 \leq i \leq n$

iii)  $A_i$  or  $B_i$  values are ordered

It may be noted that, elements of  $A_1$  may be repeated in several WCR's, unlike in the CP's.

In such a PCP all WCR's can be made equal in size by breaking the larger WCR's into several smaller ones. The hardware size of each of these PCP's can be standardized at the system level or cell level. With the WCR size fixed, each of them can be assigned a "fixed" position on the track. This reduces movement of data and eliminates need for garbage collection. The track format for this data structure is shown in Fig. 5.1. The track is divided into two halves. Each half contains one constituent of a PCP. Logically, any one of the two constituents of a PCP may be made the first constituent. This gives rise to two possible PCP's for the same binary data, one forward PCP and one reverse PCP. A header at the beginning of each half is provided for this purpose. The header contains the number of WCR's in that half, the attribute name(s) of that constituent and some flags, common to all the WCR's in that half, such as source flag and destination flag. Each standard sized WCR consists of a set of mark bits (5), an address field containing the relative position of the WCR, a value field containing a value of the first constituent and a fixed number of pointers. The pointers contain addresses of values in the second constituent of the WCR in the other half of the track. Thus all pointers in one half of the track point to addresses in the other half of the same track and excessive pointer chasing is avoided. An example of a binary relation stored as a PCP on a track is shown in Fig. 5.2.

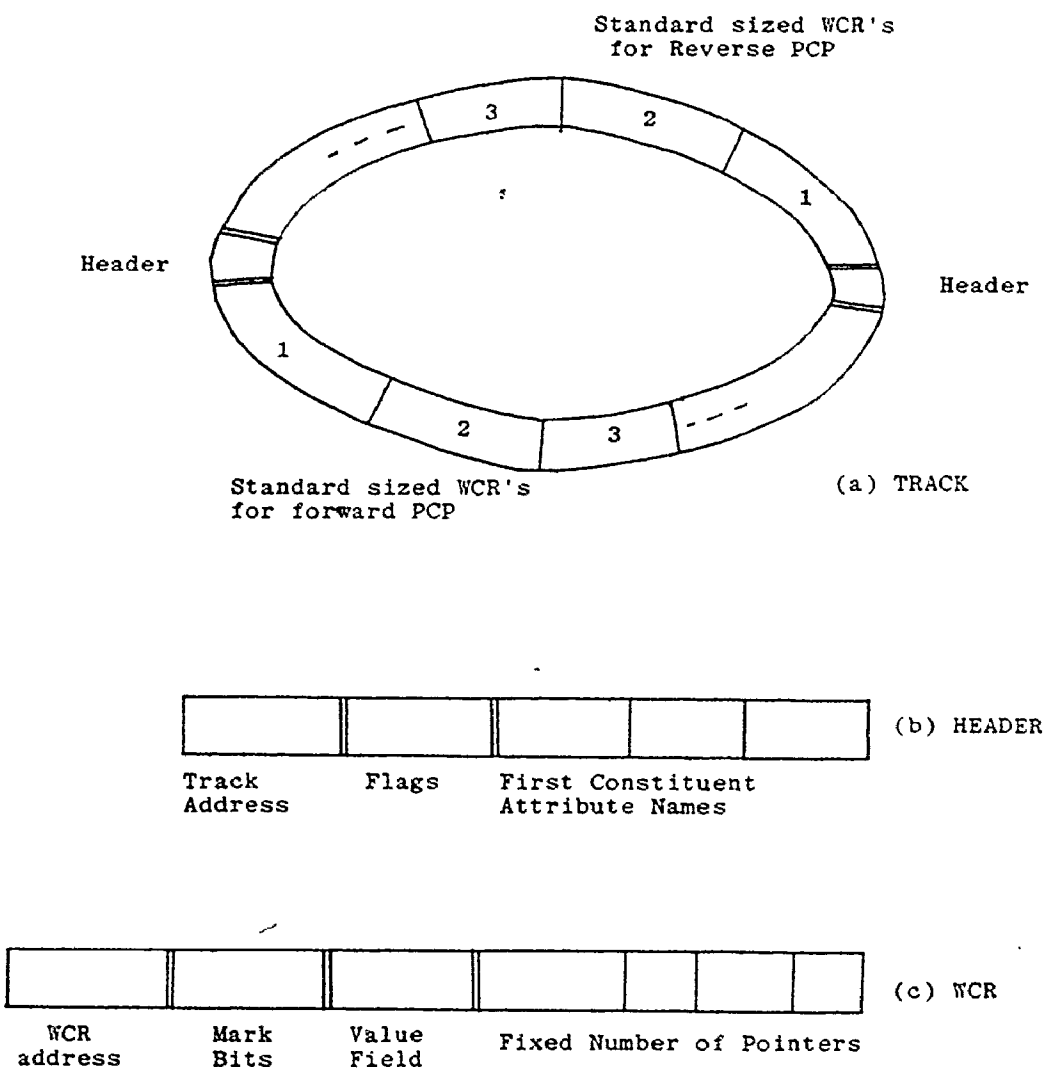


FIG. 5.1 TRACK FORMAT

$a_1$	$b_1$
$a_1$	$b_2$
$a_1$	$b_3$
$a_2$	$b_2$
$a_2$	$b_1$
$a_3$	$b_1$
$a_3$	$b_2$

## (a) Binary Relation

(1) ( , , , ) ( $a_1$ ) (1, 2, 3)(2) ( , , , ) ( $a_2$ ) (1, 2,  $\phi$ )(3) ( , , , ) ( $a_3$ ) (1, 2,  $\phi$ )

## (b) One half of track

(1) ( , , , ) ( $b_1$ ) (1, 2, 3)(2) ( , , , ) ( $b_2$ ) (1, 2, 3)(3) ( , , , ) ( $b_3$ ) (1,  $\phi$ ,  $\phi$ )

## (c) Other half of track

FIG. 5.2 PCP's on a Track - Option I

This storage structure has the following advantages. The pointers in each half of a track are always forward pointing. Therefore, there is no backward referencing and the hardware for implementation would be simpler. The data is organized in such a way that the queries based on both forward and reverse PCP's can be answered with equal ease. It also avoids duplication of data values. However, if a PCP overflows one track, it may give rise to duplication of some values at these cell boundaries. If the standard WCR size on the track is chosen in such a way that the "average" WCR size in the PCP is an integral multiple of this, the extra storage for the duplicate values and the pointers may be minimized. Also, if we use a counter to locate the WCR's (standard sized) on the track, there is no need to store the WCR address with each WCR.

#### 5.2.2 PCP - Option II

Though the use of pointers in the PCP - option I provides an efficient utilization of memory space, the pointer chasing would increase either the complexity of hardware or computation time to perform operations like join, which require data contiguity. In view of this, another data structure is presented here, which involves no pointers.

This second type of data structure exploits an inherent characteristic of the E-R model. As mentioned earlier, in all the CP's corresponding to entities and relationships, the first constituent is always a key or a system defined key. Therefore, a CP

would always be a set of EWCR's, where each non-key value may be connected to several key values, but not the other way around. Such a CP may be viewed as a set of disjoint EWCR's. It is formally defined as follows. It is different from a PCP - option I in that the elements of the constituents are not ordered.

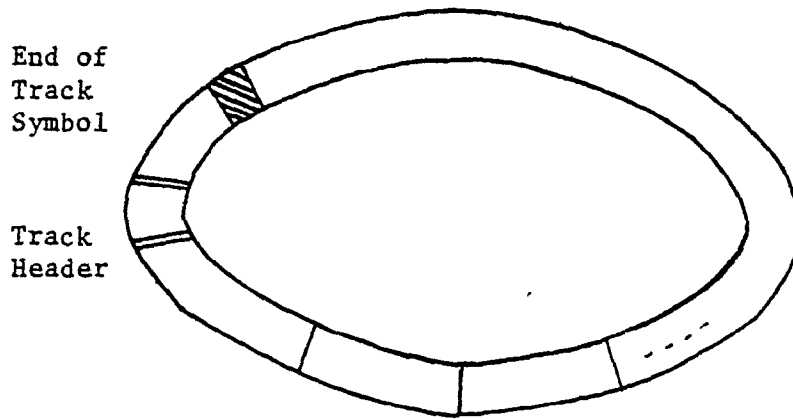
A partition of a binary relation  $R[A,B]$  is a pseudo canonical partition - option II if

$$R[A,B] = \sum_{i=1}^n w_i[A_i;B_i]$$

- where, i)  $w_i[A_i;B_i]$  is a WCR for  $1 \leq i \leq n$
- ii)  $A_i$  is a set with a single element for  $1 \leq i \leq n$
- iii) The elements of A and B are unordered.

In option II, the EWCR's may be of arbitrarily large size (number of tuples). The EWCR's can be made equal in size by breaking the larger EWCR's into several smaller ones. Now, the hardware size of each EWCR can be standardized similar to the PCP's of option I. The track format of this data structure is shown in Fig. 5.3. The track is continuous and undivided, unlike the option I format. The header of the track contains the information about track address, size of the EWCR's in the track and some flags common to the whole track. It also contains the attribute names of the first and the second constituents. Each EWCR consists of several mark bits, a first constituent value field and certain number of second constituent value fields. The number of second constituent value fields

Fig. 5.3 Track Format of PCP - Option II



(a) Track



Track Address	Size of EWCR	Flags	First Constituent Attribute Names	Second Constituent Attribute Names
---------------	--------------	-------	-----------------------------------	------------------------------------

(b) Header



Mark Bits	Nonkey Value Field (First Constituent)	Key Value Fields (Second Constituent)
-----------	--	---------------------------------------

(c) EWCR

(key values) per EWCR is standardized at the system level or track level. This makes the size of the EWCR fixed and it is specified in the track header. A hardware limit is placed on this size to facilitate hardware implementation. Thus the EWCR size may be variable within this hardware limit. An example of a binary canonical partition stored as PCP's of option II on a track is shown in Fig. 5.4.

This data structure offers the following advantages. There is no duplication of value fields as long as the EWCR size is not greater than the hardware limit. There are no pointers and hence storage requirements for pointers and pointer chasing are avoided. It is possible to view the PCP's as both forward and reverse PCP's for the same stored binary data because the second constituent values are key values and are never duplicated. Finally, the contiguity of first and second constituent values in an EWCR makes it easier to implement operations such as join and sort.


### 5.3 A Comparison of Options I and II

The storage requirements of the two proposed data structures are compared with the aid of an example. Consider a typical segment of a data base consisting of two entities  $E_1$  and  $E_2$ , connected by a relationship  $R$ , as shown in Fig. 5.5. Let  $A_1, A_2, \dots, A_{N_1}; B_1, B_2, \dots, B_3, \dots, B_{N_2}; C_1, C_2, \dots, C_{N_3}$  be the attributes of  $E_1, E_2$ , and  $R$ , respectively. The corresponding data would be stored as the following PCP's;  $[K_1; A_1], [K_1; A_2], \dots, [K_1; A_{N_1}]; [K_2; B_1], \dots, [K_2; B_{N_2}];$



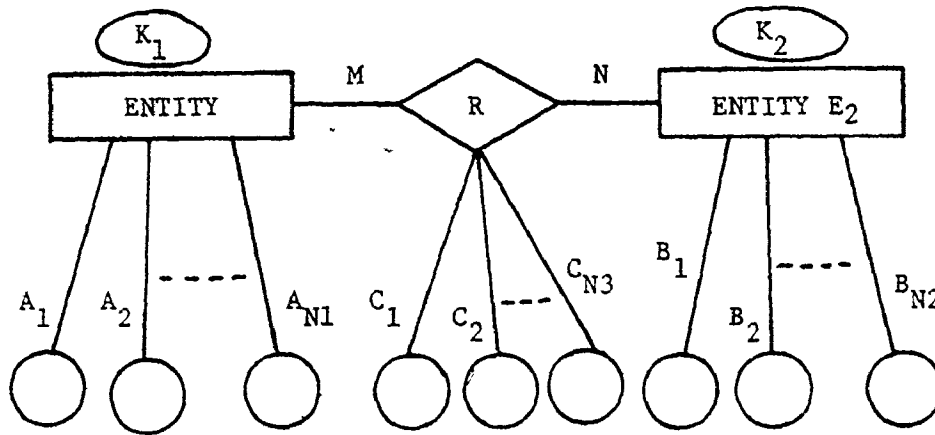
<u>Non Key Field</u>	<u>Key Field</u>
$a_1$	$b_1$
$a_1$	$b_2$
$a_2$	$b_6$
$a_2$	$b_7$
$a_2$	$b_8$
$a_3$	$b_3$
$a_3$	$b_4$

(a) A Binary Relation

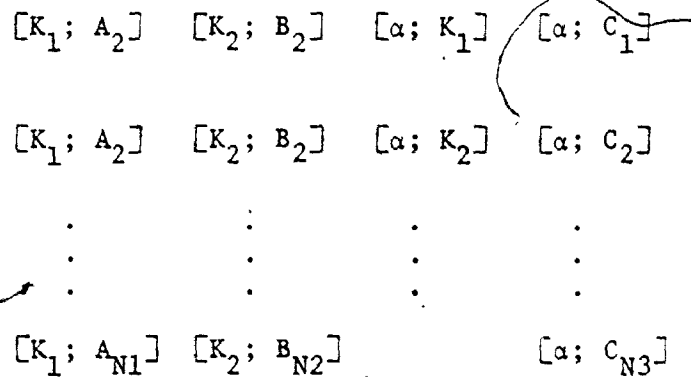
- 1) ( , , , ) ( $a_1$ ) ( $b_1, b_2, \phi$ )
  - 2) ( , , , ) ( $a_2$ ) ( $b_6, b_7, b_8$ )
  - 3) ( , , , ) ( $a_3$ ) ( $b_3, b_4, \phi$ )
-   
 Mark bits

(b) The PCP's on Track

Fig. 5.4 PCP's on a Track - Option II



(a) A Typical Segment in the Data Base



(b) The PCP's

Fig. 5.5 An Example Showing PCP's Corresponding to an E-R Schema

$[K_1; \alpha], [K_2; \alpha], [\alpha; C_1], \dots, [\alpha; C_{N_3}]$ , where  $\alpha$  is a system defined key. If the type of the relationship is 1:N, then the system defined key would be the same as  $K_2$  and the total number of PCP's would be one less. The following assumptions are made while comparing the PCP storage structures, options I and II: i) The WCR's in both halves of the tracks in PCP-option I do not have the WCR address field, ii) the values in PCP - option II are unordered, and iii) the keys are stored in coded format in both options.

Let  $m$  be the average size of an EWCR in a PCP,

$a$  be the storage for one value item in bits,

$l_1, l_2, l_3$  be the number of tuples in relations corresponding to  $E_1, E_2$  and  $R$ ,

$P_1, P_2, P_3$  be the storage need for a pointer in PCP - option I corresponding to  $E_1, E_2$  and  $R$ , respectively (in bits)

Now, let  $S_1$  be the storage required when the data is stored as PCP's of option I

$S_2$  be the storage required when the same data is stored as PCP's of option II.

The total number of PCP's  $N = N_1 + N_2 + N_3 + 2$ . The storage for PCP's corresponding to entity  $E_1$ ,  $S_{11}$  is given by:

$$\begin{aligned} S_{11} &= N_1 \cdot \{\text{storage for one PCP}\} \\ &= N_1 \cdot \{\text{storage for first constituent values} + \text{storage for} \\ &\quad \text{second constituent values} + \text{storage for pointers}\} \end{aligned}$$

$$= N_1 \cdot \left[ \ell_1 \cdot p_1 + \frac{\ell_1}{m} \cdot a + 2\ell_1 \cdot p_1 \right] \quad (5.1)$$

Note, the number of second constituent values =  $\frac{\ell_1}{m}$ , and

$$\text{the total number of pointers} = 2\ell_1$$

Similarly, storage for PCP's corresponding to entity  $E_2$ ,  $S_{12}$  is given by

$$S_{12} = N_2 \cdot \left[ \ell_2 \cdot p_2 + \frac{\ell_2}{m} \cdot a + 2\ell_2 \cdot p_2 \right] \quad (5.2)$$

The storage for PCP's corresponding to relationship R,  $S_{13}$  is given by

$$S_{13} = N_3 \cdot \left[ \ell_3 \cdot p_3 + \frac{\ell_3}{m} \cdot a + 2\ell_3 \cdot p_3 \right] \\ + \frac{\ell_1}{m} \cdot p_1 + \frac{\ell_2}{m} \cdot p_2 + 6\ell_3 \cdot p_3 \quad (5.3)$$

$$S_1 = S_{11} + S_{12} + S_{13}$$

$$\therefore S_1 = \sum_{i=1}^3 N_i \left[ \ell_i \cdot p_i + \frac{\ell_i}{m} \cdot a + 2\ell_i \cdot p_i \right] \\ + \frac{\ell_1}{m} \cdot p_1 + \frac{\ell_2}{m} \cdot p_2 + 6\ell_3 \cdot p_3 \quad (5.4)$$

But  $p_i = \lceil \log_2 \ell_i \rceil$ , because we need pointers to point only half of each track. (5.5)

$$\therefore S_1 = \sum_{i=1}^3 N_i \left\{ 3\ell_i \lceil \log_2 \ell_i \rceil + \frac{\ell_i}{m} \cdot a \right\} + \frac{1}{m} \left\{ \ell_1 \lceil \log_2 \ell_1 \rceil \right. \\ \left. + \ell_2 \lceil \log_2 \ell_2 \rceil \right\} + 6\ell_3 \lceil \log_2 \ell_3 \rceil \quad (5.6)$$

Similarly,  $S_2 = S_{21} + S_{22} + S_{23}$  where

$S_{21}$  - is the storage for PCP's of entity  $E_1$ ,

$S_{22}$  - is the storage for PCP's of entity  $E_2$

$S_{23}$  - is the storage for PCP's of relationship R

$$\begin{aligned} S_{21} &= \text{No. of PCP's} \cdot \{\text{storage for one PCP}\} \\ &= N_1 \cdot \{\text{storage for non key values} + \text{storage for key values}\} \\ &= N_1 \cdot \left\{ \frac{\ell_1}{m} \cdot a + \ell_1 \cdot \lceil \log_2 \ell_1 \rceil \right\} \end{aligned} \quad (5.7)$$

$$S_{22} = N_2 \cdot \left\{ \frac{\ell_2}{m} \cdot a + \ell_2 \lceil \log_2 \ell_2 \rceil \right\} \quad (5.8)$$

$$\begin{aligned} S_{23} &= N_3 \cdot \left\{ \frac{\ell_3}{m} \cdot a + \ell_3 \lceil \log_2 \ell_3 \rceil \right\} + \frac{\ell_1}{m} \cdot \lceil \log_2 \ell_1 \rceil + \frac{\ell_2}{m} \lceil \log_2 \ell_2 \rceil + \\ &2 \cdot \ell_3 \lceil \log_2 \ell_3 \rceil \end{aligned} \quad (5.9)$$

$$\begin{aligned} \therefore S_2 &= \sum_{i=1}^3 N_i \left\{ \ell_i \lceil \log_2 \ell_i \rceil + \frac{\ell_i}{m} \cdot a \right\} + \frac{1}{m} \left\{ \ell_1 \lceil \log_2 \ell_1 \rceil \right. \\ &\left. + \ell_2 \lceil \log_2 \ell_2 \rceil \right\} + 2\ell_3 \lceil \log_2 \ell_3 \rceil \end{aligned} \quad (5.10)$$

From Equations 5.6 and 5.10, it is easy to see that  $S_1 \gg S_2$ .

Therefore, PCP - Option II offers a better storage structure.

In the above analysis, it is assumed that the average size of an EWCR in a PCP ( $m$ ) is equal to the standard size of EWCR ( $s_t$ ) on the track. But this assumption does not hold good if some of the EWCR's have a large number of tuples. When these are broken down into smaller ones, there would be repetition of nonkey values. If  $P$  is the

proportion of large EWCR's in a PCP, the storage  $S_p$  needed for one such PCP with some repeated nonkey values is given by

$$S_p = \frac{a \cdot \ell \cdot}{m} (1-P) + \frac{a \ell}{s_t} P \quad (5.11)$$

when  $P$  is very small  $S_p \approx \frac{a \ell}{m}$

#### 5.4 WCRML Instruction Set

The WCRC system supports a machine oriented instruction set at the internal level. These instructions are implemented in hardware. The WCRL commands at the conceptual level are translated into these WCRML instructions. The WCRML instructions can be classified into the following two major types:

- i) Instructions executed by the cell processors
- ii) Instructions executed by the query processors

##### 5.4.1 Instructions Executed by the Cell Processor

The general format of all these instructions is:

<LABEL> <OPCODE> <OPERAND> <CONDITION> <PARAMETERS>

The label specifies an optional symbolic instruction address. The opcode specifies the operation to be performed. The operand indicates the cell name or address and the constituent name. The condition may take any one of the following forms:

- a) Null
- b)  $q_1 \wedge q_2 \wedge q_3 \dots \wedge q_k$

- c)  $q_1 \vee q_2 \vee q_3 \dots \vee q_k$ , where  $q_i$  is of the form,
- i)  $\langle \text{CONSTITUENT} \rangle \langle \text{OPR} \rangle \langle \text{OPERAND} \rangle$ . OPR is one of =,  $\neq$ , <, >,  $\leq$ ,  $\geq$  and OPERAND can be a constant, a register, or a constituent name.
  - ii) marked (x), denoting a combination of mark bits set.
  - iii) unmarked (x), denoting a combination of mark bits reset.
  - iv) RAM marked. (Note, each cell has a 1-bit wide RAM whose length is equal to the number of key values on the track).

This qualification is valid only for a key constituent. It is 'true' if the corresponding RAM bit is set. Within a cell, the qualifications are either in a conjunction or disjunction. But a selection clause over the cells may be a combination of conjunctions and disjunctions of these conditions.  $k$  denotes the maximum number of comparators in the cell. In option I, the OPERAND may be restricted to be only a constant or a register, otherwise qualification evaluation would take several revolutions. However, in option II, the OPERAND may be either a constant, a register, or the other constituent on the cell. Also note, null values of the constituents would not qualify for any condition, i.e., they would return 'False' to every condition.

The cell executed instructions can be divided into the following types:

- a) Retrieval
- b) Set function
- c) Update
- d) Storage definition

Most of these instructions are executable in one or two revolutions of the cells. These instructions are supplied by the query processor to cell processors. The instructions are described below:

a) Retrieval Instructions:

1) SET

<LABEL> SET (M) [<PCP name>:<constituents>](CONDITION) (X,Y,N,Z)

This instruction sets the mark bit (M) of the qualified constituent(s) values of EWCR's on the track. The mark bits of unqualified values are left untouched. The parameter X (All, First) specifies whether all/first of the qualified values are to be set; Y (Yes, No) specifies whether the other constituent values that go with the marked values have to be set or not. N indicates the mark bit of the other constituent. Z (Yes, No) indicates whether the RAM associated with the cell has to be set or not. Recall, each cell of a PCP contains a 1 bit wide RAM whose length is equal to the number of key values stored on the track. Similarly, each query processor also contains a 1 bit wide RAM whose length is equal to maximum size of cell RAM's. (The details of the cell and query processor hard-



ware are discussed later). A QP can read any cell RAM into its own RAM.

2) RESET

<LABEL> RESET (M) [<PCP name>:<constituent>]

This instruction resets the mark bits of the constituent(s) specified. It does exactly the opposite of SET.

3) READ

<LABEL> READ [<Cell#/PCP Name> / <constituent(s)>](CONDITION)  
(BA/R, X, Y)

This instruction reads the qualified values of the constituent(s) into the buffer area (BA) specified. The option to reset, after reading is provided by parameter Y (Yes, No). The parameter X specified whether All or just One value have to be read from the cell/PCP involved. When only one value is being read, it may be read into a register specified by parameter R.

b) Set Function Instructions

5) MIN

<LABEL> MIN [<PCP name>:<constituent>] (CONDITION)  
(R#)

This instruction obtains the minimum of the marked values of the constituent specified into the register R# specified. Each of the cells belonging to the PCP compute their local minima and store them in the specified register. If no register is specified,

the result is palced in a special register called the result register of the cell. The global minimum is computed by the query processors using a set function instruction (explained later). All the registers in a cell are accessible to the QP.

Note, the CONDITION here does not need the RAM marked option.

6) MAX

```
<LABEL> MAX [<PCP name>:<constituent>] (CONDITION) (R#)
```

This instruction is similar to MIN, except the register would contain the maximum value.

7) SUM

```
<LABEL> SUM [<PCP Name>:<constituent>] (CONDITION) (R#)
```

This instruction computes the sum of all distinct qualified values of the constituent specified and stores it in the register specified. The overall sum is computed by QP. Again, CONDITION does not need RAM marked option.

8) CAR

```
<LABEL> CAR <PCP name>:<constituent> (CONDITION) (R#)
```

Distinct qualified values of the key are counted and the result is stored into the register indicated. The overall result is obtained by QP.

9) AVG

```
<LABEL> AVG [<PCP name>:<constituent>] (CONDITION) (R#)
```

This instruction computes SUM of a nonkey attribute and CAR of the corresponding key attribute and divides SUM by CAR to obtain AVG and stores it in the register specified. The overall result is computed by QP.

c) Update Instructions:

10) INSERT

<LABEL> INSERT [<PCP name>](<Buffer Area>/<tuple>)

This instruction inserts the EWCR's in the buffer area into the first available track of the PCP specified. If no track is available, then a new cell is formatted and insertion is carried out.

11) DELETE

<LABEL> DELETE [<PCP name>:<constituent(s)>] (CONDITION) (X)

This instruction deletes the qualified values of the specified constituent(s). i.e. It replaces the value by null. If only one constituent is specified, the parameter X allows the option to delete the values of the other constituent connected to a qualified value.

X can be Yes or No.

12) MODIFY

<LABEL> MODIFY [<PCP name>:<constituent(s)>] (CONDITION)  
<OPD(s)>)

This instruction replaces the qualified values of the constituent(s) by the opd(s) where opd can be a constant or a register.

## 13) ADD

<LABEL> ADD [<PCP name>:<constituent>] (CONDITION) (<opd>)

This instruction adds opd to the qualified values of the specified constituent. opd can be a constant or a register. The following instructions are similar to ADD except that they carry a different operation specified by the opcode.

## 14) SUB

<LABEL> SUB [<PCP name>:<constituent>] (CONDITION) (<opd>)

## 15) MUL

<LABEL> MUL [<PCP name>:<constituent>] (CONDITION) (<opd>)

## 16) DIV

<LABEL> DIV [<PCP name>:<constituent>] (CONDITION) (<opd>)

## 17) LOAD

<LABEL> LOAD [<cell address>] (Reg 1, <opd>)

This instruction allows for register transfer between the registers of the specified cell. If <opd> is a constant, it would be loaded directly into Reg 1.

d) Storage Definition Instructions

18) <LABEL> CREATE [<PCP name>:<cell#>] (<Format>)

This instruction creates a new track and formats it according

to the data supplied in parameter 'Format'. The query will supply the cell processor with the format data. The format specifies the header(s) and a typical tuple(s) on the track. This information may be specified along with the instruction or in the work area, the address of which is specified by the instruction. The new cell is given the specified name and the entire track is formatted to receive tuples of type specified (i.e. gaps and headers are stored on the track).

19) DESTROY [<PCP name>:<cell#>]

This instruction deletes all the EWCR's and some of the header information of the cell/PCP specified. If the cell# is not specified, the whole PCP is destroyed. A flag is set at the beginning of the track indicating that the cell is empty. Except for the flag bits and the track address, all the other header information is destroyed.

#### 5.4.2 Instructions Executed by a Query Processor

These instructions can be divided into five major types:

- a) Inter-cell communication instructions
- b) Set function instruction
- c) Program control instruction
- d) I/O instruction
- e) End of program instruction

a) Inter-Cell Communication Instructions

As mentioned earlier, cells have no direct communication between them. Usually, there is very little transfer of information between cells. Therefore, the hardware provision for direct inter-cell communication cannot be justified in terms of cost and complexity. However, indirect communication is provided through the QP by means of a set of instructions executable by QP. The general format of these instructions is as follows:

<LABEL> <OPCODE> [<SOURCE>] <DESTINATION>] (<parameters>)

The opcode specifies the operation to be performed, the source indicates the source PCP or cell from where the information is transferred. The destination indicates the cell or PCP which is receiving this information through the QP. The details of the instructions are given below.

1) <LABEL> TRANS\_RAM [<source PCP cell>] [<Dest. PCP/cell>]

This instruction transfers the contents of the RAM of a source cell into the RAM of destination cell or the RAM POOL of a source PCP into RAM POOL of destination PCP. The RAMS of all tracks corresponding to a PCP put together, are called a RAM POOL. Note, the source and destination RAM's/RAM POOL's should be compatible in size.

2) <LABEL> BOP\_MARK [<Source PCP list>] [<Dest. PCP list>]

(BW)

This instruction performs boolean operation indicated by the boolean word (BW) on the RAM POOLS of the source PCP's and the result on to the RAM POOL(S) of the destination PCP(s). The BW allows specification of the combination of boolean operations on the source PCP's. It is a word of ones and zeroes where a one indicates AND and zero indicates an OR.

3) <LABEL> TRANS\_REG (<Source cell/Source PCP>)  
(<Dest.cell/QP/PCP>) (SR#, DR#)

This instruction transfers the contents of the specified register from the source cell (SR#) into the register of the destination cell (DR#). The destination can also be the QP.

b) Set Function Instruction

This instruction is of the following format.

4) <LABEL> SET FUNC (<source PCP>) (<sopr>)

This instruction reads the local set function values from the result registers of the PCP specified, and computes the global value and stores it in a register called SF Register in the query processor. <sopr> specifies the type of the set function (MIN, MAX, AVG, CAR, SUM).

c) Program Control Instruction

5) <LABEL> BRANCH (LABEL/Address) (condition) (DEC/INC)

This instruction is used for the program control. It can

work both as conditional or unconditional branch instruction. The condition may be of the form  $\langle \text{Reg} \rangle \langle \text{opr} \rangle \langle \text{opd} \rangle$ , where opr is one of ( $=, \neq, >, <, \geq, \leq$ ) and opd is a constant. Before the condition is tested, the Reg may be incremented or decremented by one (INC/DEC).

d) I/O Instruction

6)  $\langle \text{LABEL} \rangle \text{ JOIN } (\langle \text{PCP list} \rangle) (\text{NJ/CP, key PCP, SA})$

This instruction performs the join over the marked key values of the PCP's specified. Note, this join can be a natural join or a cross product (NJ/CP). The parameter SA specifies the attribute over which the resultant tuples should be sorted.

e) End of Program Instruction

7)  $\langle \text{LABEL} \rangle \text{ END}$

This instruction indicates the end of a WCRML program.

5.4.3 Translation of WCRL into WCRML Instructions

In this section, the translation of WCRL commands into WCRML instructions is demonstrated with the aid of some examples.

Consider the data base shown in Fig. 5.6a, involving Employees, Dept., Company and Item. The corresponding PCP's are shown in Fig.

5.6 b. Note,  $\alpha_1$  is a system defined key.

Example Queries

Q1) Find the employees (E#) whose salary is greater than that of



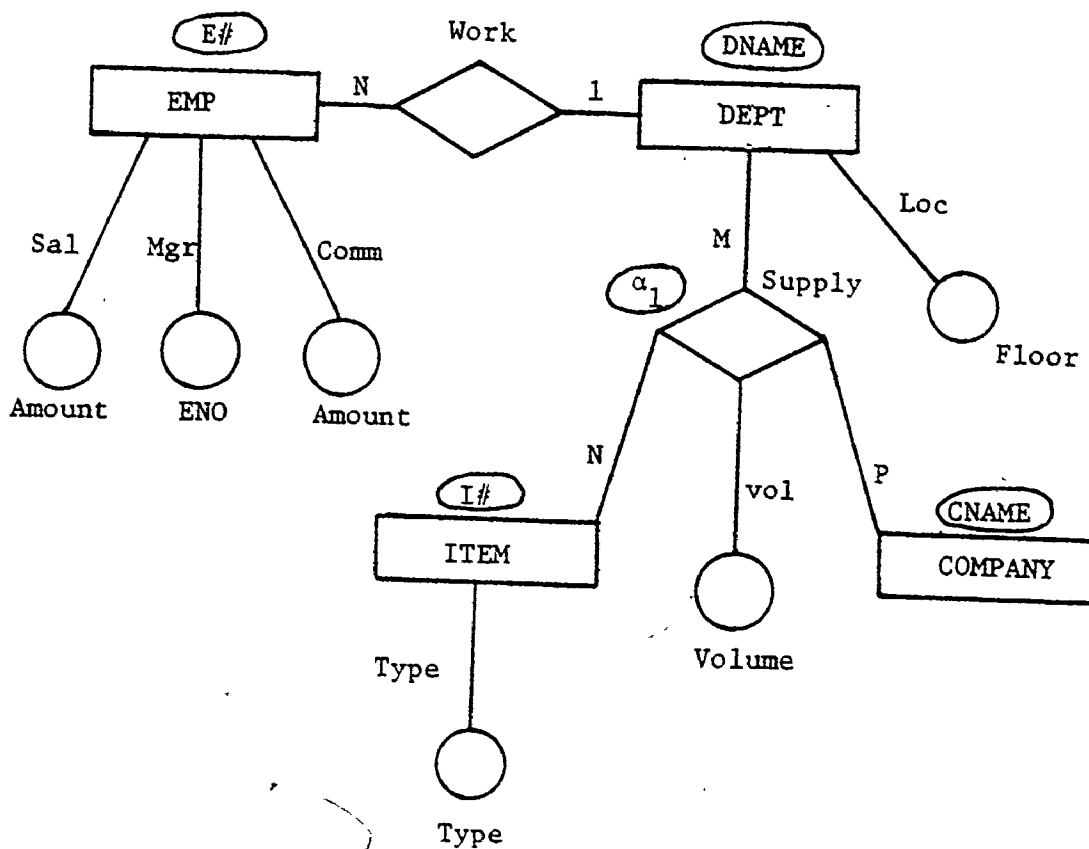


Fig. 5.6 a An Example Data Base

PCP's:

- |                                    |  |
|------------------------------------|--|
| $P_1$ [ $\underline{E\#}$ ; SAL]   | $P_6$ [ $I\#$ ; TYPE]                    |
| $P_2$ [ $\underline{E\#}$ ; MGR]   | $P_7$ [ $\underline{\alpha_1}$ ; DNAME]  |
| $P_3$ [ $\underline{E\#}$ ; COMM]  | $P_8$ [ $\underline{\alpha_1}$ ; I#]     |
| $P_4$ [ $\underline{E\#}$ ; DNAME] | $P_9$ [ $\underline{\alpha_1}$ ; CNAME]  |
| $P_5$ [ $\underline{DNAME}$ ; LOC] | $P_{10}$ [ $\underline{\alpha_1}$ ; VOL] |

Fig. 5.6 b The PCP's

any employee in the 'AUTO' department WCRL:

$$\sigma_{sr} \{ \sigma_{cj} p_1 [E\#; SAL], p_4 [E\#; Dname] (p_1 \cdot E3 = p_4 \cdot E\#)$$

o

$$(p_4 \cdot Dname = AUTO)) \} (\phi; SAL) = c_1[\phi; SAL]$$

$$\sigma_{sr} \{ \sigma_{cj} p_1 [E\#; SAL], c_1[\phi; SAL] (p_1 \cdot SAL > c_1 \cdot SAL) \}$$

$$(E\#; \phi) = c_2 [E\#; \phi]$$

#### WCRL

SET (A) [P<sub>4</sub>: Dname] (DName = AUTO) (All, Yes,  $\phi$ , Yes)

TRANS\_RAM [P<sub>4</sub>] [P<sub>1</sub>]

SET (A) [P<sub>1</sub>: E#] (RAM Marked) (All, Yes, A, No)

MAX [P<sub>1</sub>: SAL] (Marked (A)) (R<sub>1</sub>)

SET (B) [P<sub>1</sub>: SAL] (SAL R<sub>1</sub>) (All, Yes, B, No)

READ [P<sub>1</sub>: E#] (Marked (B)) (Buffer, All, Yes)

Now the Buffer contains all the E#'s who satisfy the query condition.

Q2) List the E#'s and their managers and their salaries who work in the department 'AUTO' and whose salary is greater than their annual commission. (Note, this query has an inter-

attribute selection clause).

WCRL

$$\sigma_{s1} \{ \sigma_{cj} P_4 [E\#, DNAME] P_1 [E\#; SAL] P_3 [E\#; COMM]$$

$$((P_4 \cdot E\# = P_1 \cdot E\#) \quad (P_1 \cdot E\# = P_3 \cdot E\#))$$

$$(P_4 \cdot DNAME = 'AUTO') \quad (P_1 \cdot SAL > P_3 \cdot COMM)$$

$$(E\#; SAL) = C_1 [E\#; \phi]$$

$$\sigma_{sr} \{ \sigma_{cj} P_2 [E\#; MGR] C_1 [E\#; \phi] (P_2 \cdot E\# = C_1 \cdot E\#) \}$$

$$(E\#; Mgr) = C_2 [E\#; MGR]$$

WCRML

SET (A) [P<sub>4</sub>:DNAME] (DNAME=AUTO) (ALL, Yes, A, Yes)

TRANS\_RAM [P<sub>4</sub>] [P<sub>1</sub>]

SET (A) [P<sub>1</sub>:E#] (RAM Marked) (ALL, Yes, A, Yes)

CAR [P<sub>1</sub>:SAL]

SET\_FUNC [P<sub>1</sub>] (R $\phi$ ) (R is a QP register)

LOOP READ [P<sub>1</sub>:SAL] (Marked (A)) (R<sub>1</sub>, One, Yes)

TRANS\_REG [P<sub>1</sub>] [P<sub>3</sub>] (R<sub>1</sub>, R<sub>2</sub>)

SET (B) [P<sub>3</sub>:COMM] (COMM < R<sub>2</sub>) (ALL, Yes, B, Yes)

TRANS\_RAM [P<sub>3</sub>] [P<sub>1</sub>, P<sub>2</sub>]

READ [P<sub>2</sub>:(E#, MGR)] (RAM Marked) (BA, ALL, Yes)

READ [P<sub>1</sub>:(E#, SAL)] (RAM Marked) (BA, ALL, Yes)

JOIN [P<sub>1</sub>, P<sub>2</sub>] (NJ, P<sub>1</sub>:E#,  $\phi$ )

BRANCH (LOOP) (R $\phi$  > 0) (DEC)

Q3) Move the location of 'CRANE' department to 2nd floor

WCRL

$\sigma_M$  DEPT [Entity](DNAME = 'CRANE') ( , 2)

Note: A blank in the tuple indicates that the value is to remain unchanged and a  $\phi$  indicates the value to be changed to null

WCRML

MODIFY [P<sub>5</sub>; LOC] (Dname = 'Crane') ( 2 )

Q4) Insert an Employee record with E#, Salary, Mgr#, Commission,  
as follows:

"2020, 10k, 64, 2k"

WCRL

$\sigma_I$  EMP [Entity] ("2020, 10k, 64, 2k")

WCRML

INSERT [P<sub>1</sub>] ('2020, 10k')

INSERT [P<sub>2</sub>] ('2020, 64')

INSERT [P<sub>3</sub>] ('2020, 2k')

### 5.5 Cell Hardware

In this section, the details of cell hardware are presented to a moderate level of complexity. The overall functional organization of a cell is shown in Fig. 5.7. It consists of the following seven major blocks:

- a) Circulating memory or track (CM)
- b) Track format identification unit (TFIU)
- c) Buffer unit (BU)
- d) Search unit (SU)
- e) Arithmetic and logic unit (ALU)
- f) RAM logic unit (RLU)
- g) I/O Unit (IOU)
- h) Control unit (CU)

#### 5.5.1 Circulating Memory (CM)

Each cell has a circulating sequential memory (track). CCD memory technology or any other current technology which provides low cost per bit and high bit rate should be chosen for implementing this

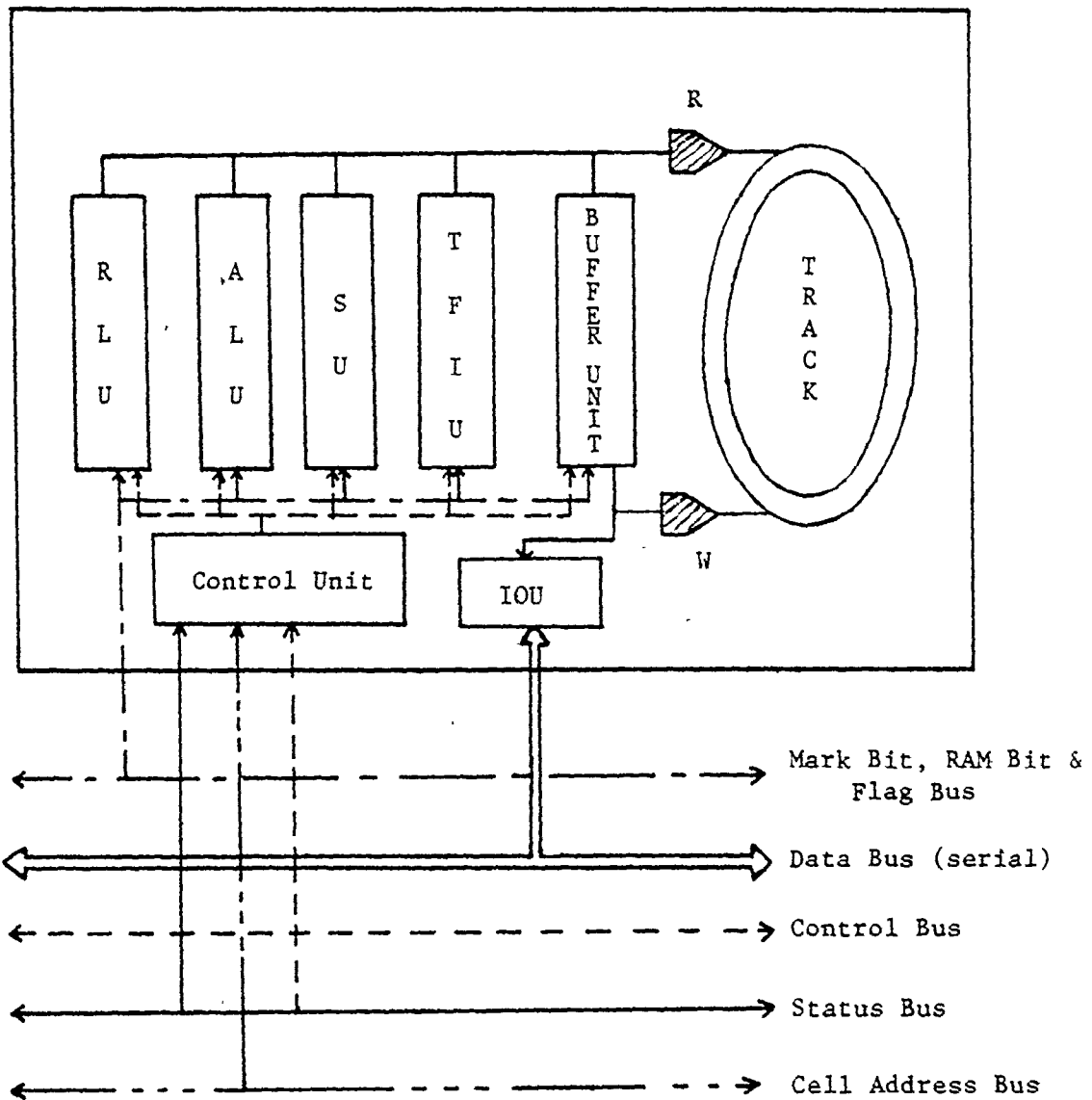
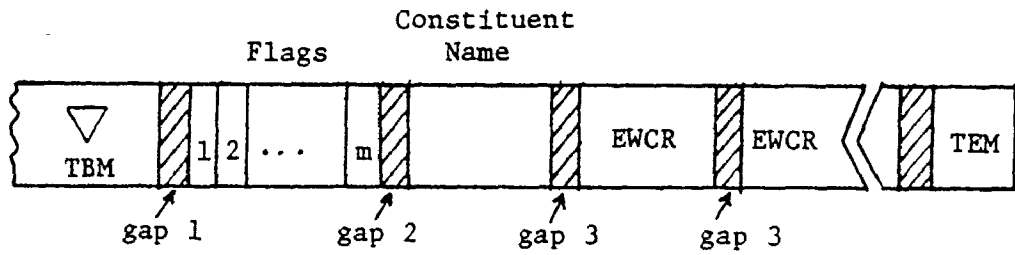


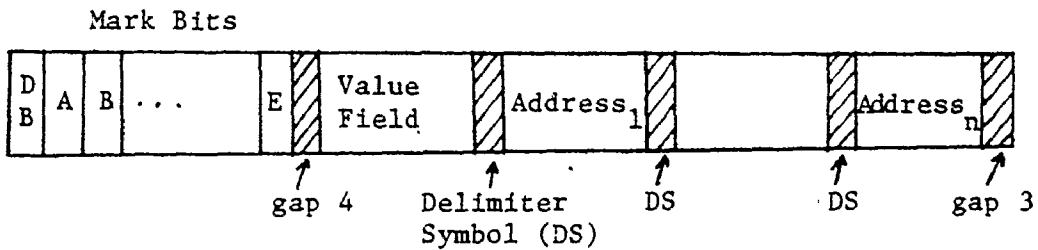
Fig. 5.7 Overall Organization of a Cell

memory. The data is read and written via a fixed head pair. The time taken to read the whole track contents is called one revolution time ( $t_r$ ).

The data is stored as PCP's on the tracks. Each track consists of several EWCR's which are laid out with gaps in between. These gaps provide the time required to issue the control signals for the operations on the data when it passes through the buffer unit. The storage formats for option I and option II are shown in Fig. 5.8. For illustration, the size of an EWCR is assumed to be 5 binary tuples. In practice, this is a system parameter ( $s_t$ ), which has to be chosen based on the nature of the data. Each track has a track beginning marker symbol (TBM) and a track end marker (TEM). TEM indicates the logical end of the track. Note, in option 1, both halves have the same format for the header and the number of address fields ( $n$ ) is 1 for the first half and 5 for the second. All the fields except value fields, have a fixed length defined at the system level. The length of value field may be 2, 4, 6, or 8 bytes depending on the type of data. This may vary from cell to cell, but within a cell all the value fields of a constituent are of the same length. Thus the nonkey value field has a fixed length defined at cell level. The key value fields are stored in a coded format and hence a maximum 2 bytes would be sufficient for storing them. A nonkey value field may be an integer number or a character string. For option I, the value fields for the keys have an ordering on them, which eliminates the need for storing the PCP address. If ordering is not maintained, then an extra PCP address



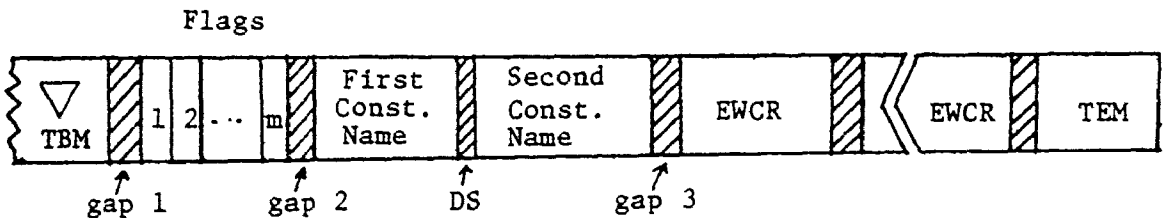
i) Header (same for both Halves)



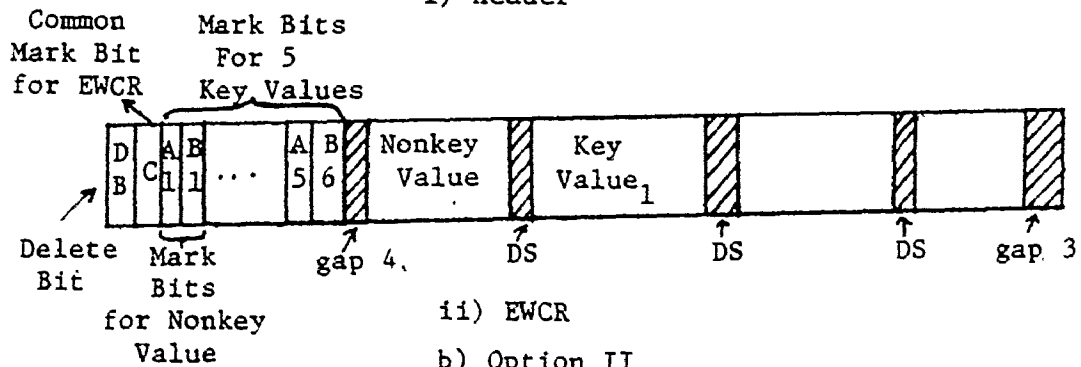
ii) EWCR

Note n = 1 for key constituent (first half track)  
 n = 5 for Nonkey constituent (second half track)

a) Option I



i) Header



ii) EWCR

b) Option II

Fig. 5.8 Storage Format



field has to be introduced in each EWCR just before the mark bits, whose length would be  $\lceil \log_2 P \rceil$ , where P is the number of EWCR's in the other half of the track.

In option II, a 'directory' track for each PCP may be maintained at the internal level which stores the actual key values and their encoded values.

#### 5.5.2 Track Format Identification Unit (TFIU)

This unit detects the format of the track and sets on the appropriate indicators to control the operations in the cell. The organization of this unit consists of the following subunits:

- a) Gap Indicators
- b) TEM, TBM and DS Indicators
- c) Bit sequence indicators
- d) Flag indicators
- e) Constituent name indicator
- f) Mark bit indicators

a) Gap Indicators: There are four counters - one for each gap type to detect the gaps. Before any operation is performed on a cell, a 'sync' signal is issued by the QP processor to the cell. This signal is used to align beginning of the track under the read-head. The gap lengths are chosen in such a way that they allow enough time for performing the operations. A typical circuit for detecting these gaps is shown in Fig. 5.9. Each detector has a

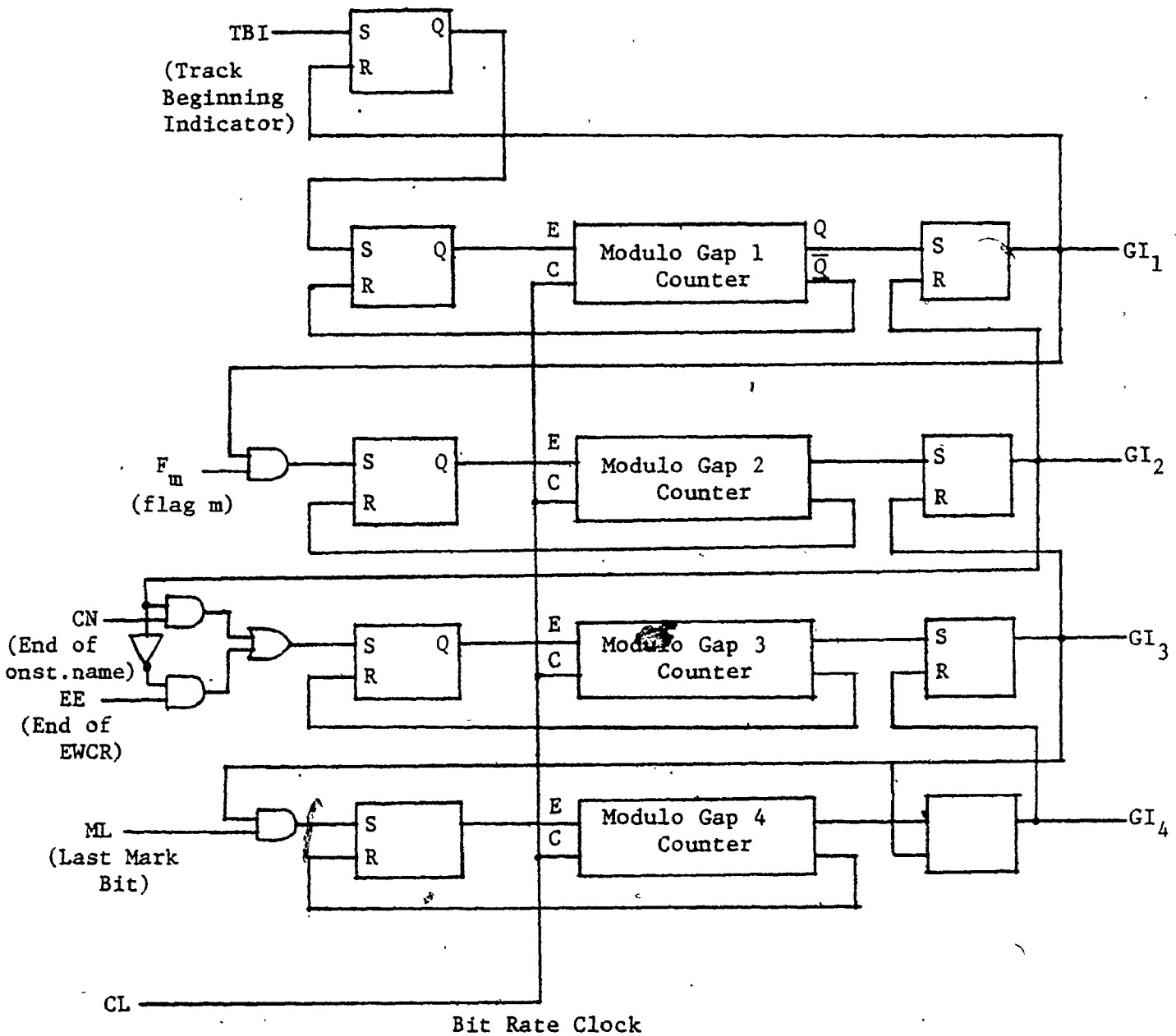


Fig. 5.9 Gap Indicators

Modulo-'gap length' counter, which starts counting the number of bits after proper control lines (such as TBI from the TBM Indicator) are set. At the end of the count, that particular indicator is set on and it is reset when the next gap indicator is set.

The signals TBI,  $F_m$ , CN AND  $M_L$  are obtained from TBM indicator, flag indicator (m), constituent name indicator and mark bit indicators, respectively as described below.

b) TEM, TBM and DS Indicators: The encoded patterns of these marker symbols are stored in three sequential registers. These patterns are compared on a bit serial basis with the incoming data from the read-head at the appropriate time. The hardware to perform the comparisons is shown in Fig. 5.10. The timing is controlled by the gap indicators and other control signals such as EF (end of value or address field). The delay D is required to reset DSI signal so that the next Delimiter symbol is recognized again.

c) Bit Sequence Indicator: This hardware generates sequences synchronized to the track bit rate starting from the end of gaps. For example, the six bit sequences after gap 3 (in option I) would indicate the six mark bit positions on the track. These sequences are decoded and used to check the incoming data at the appropriate time for mark bits. Note, the gaps denote the beginning of record blocks such as value fields, mark bit field, etc. Correspondingly, there are five bit sequence generators (gap 1-4 and DS). Fig. 5.11 a and b show two such bit sequence indicators. The first one is used

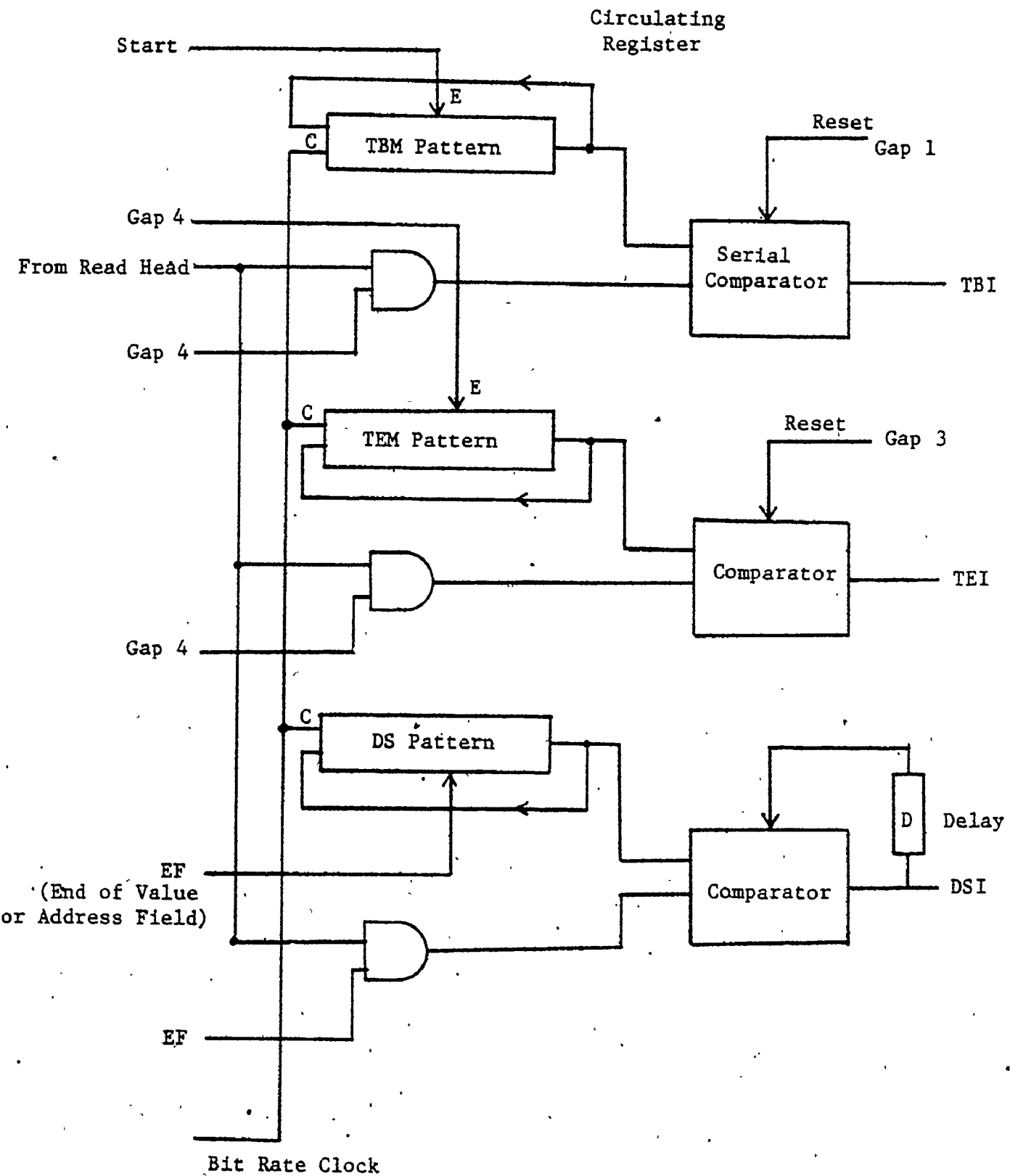


Fig. 5.10 . TEM, TBM and DS Indicators

for flags and mark bits. The second one is used for indicating beginning and end of a field.

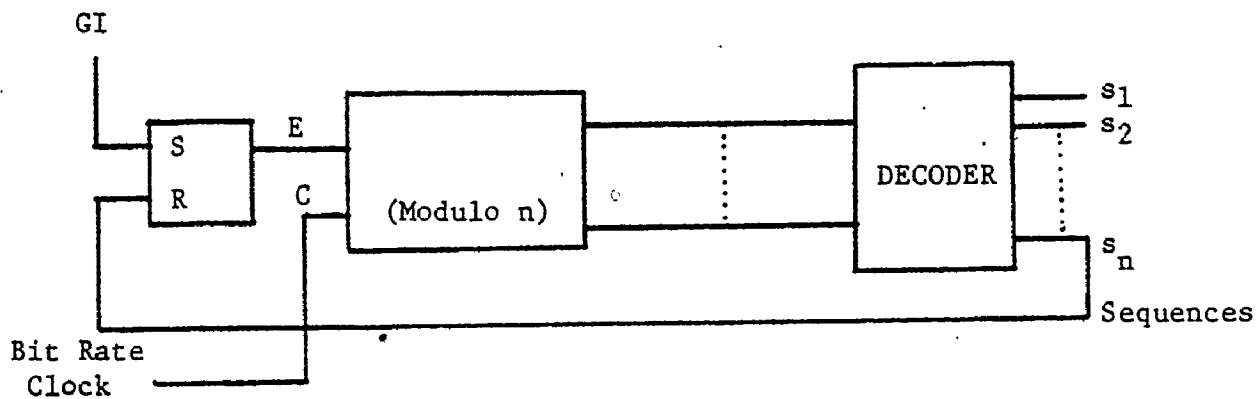
d) Flag Indicators: These are just flip flops which indicate the status of the corresponding flag of the track header. One such flag indicator is shown in Fig. 5.11 c.

e) Constituent Name Indicator: This circuit is similar to bit sequence indicator. It indicates the starting and ending of constituent name field. These indicator signals are used by search unit to load the constituent name from the read head into one of its comparators for constituent name evaluation.

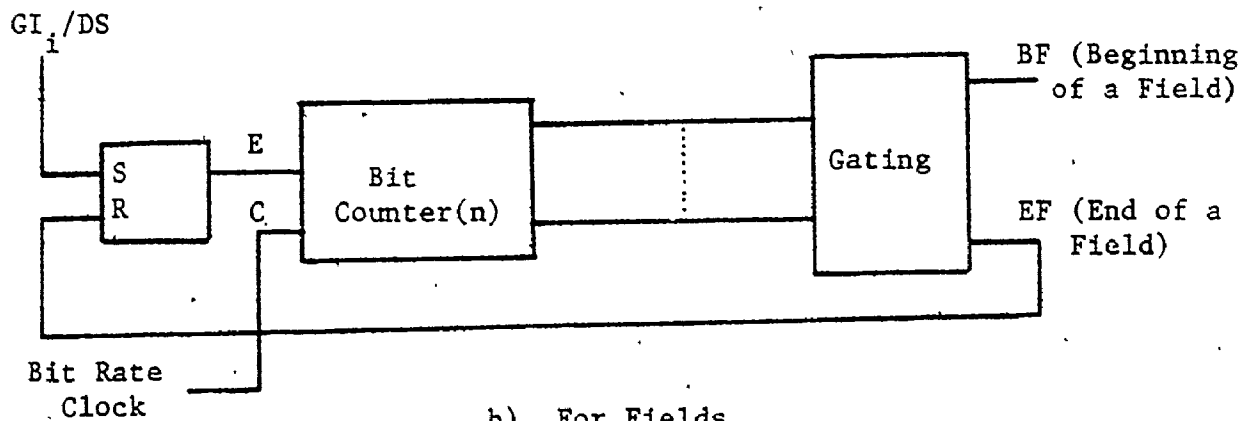
f) Mark Bit Indicators: These are similar to flag indicators. Each mark bit has an associated flip flop, which is set or reset based on the incoming data at the appropriate time. The timing is provided by the bit sequence indicators.

### 5.5.3 Buffer Unit (BU)

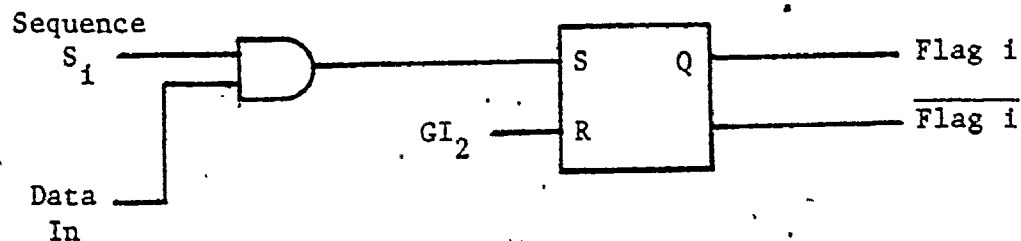
The buffer unit consists of a buffer register and an auxiliary register. The length of the buffer register is chosen in such a way that it can hold an EWCR and gap 3. It provides the necessary delay, between read and write, which will be used to perform the operations. The auxiliary register is used for insert and update operations. If the EWCR passing through the buffer is not a qualified EWCR, it is written back as it is on to the track. Output of the buffer passes



a) For Mark Bits



b) For Fields



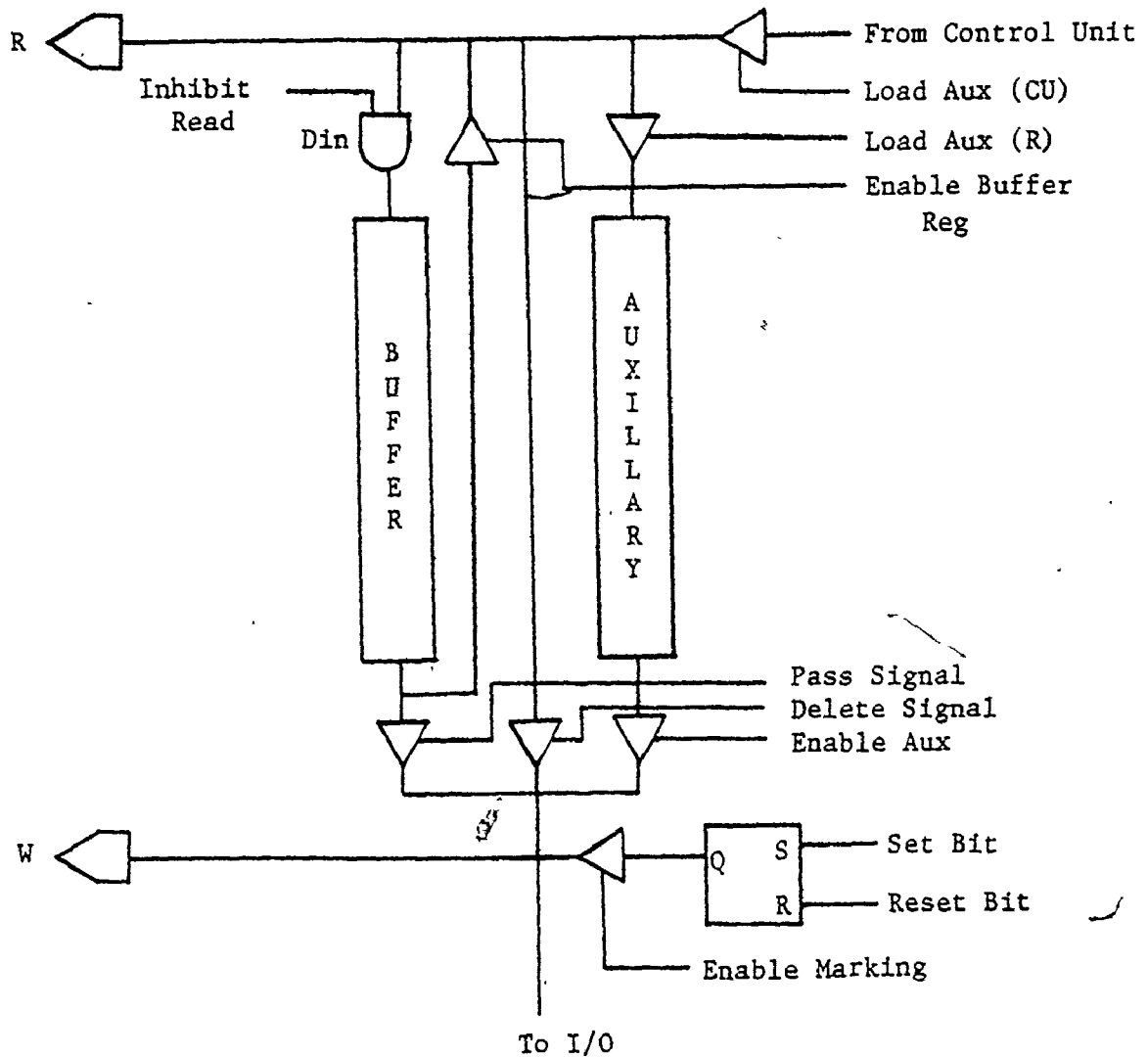
c) Flag Indicator

Fig. 5.11 Bit Sequence Indicators and Flag Indicators

through write head which enables setting or resetting of the flags or mark bits. Changes in the value or address fields are affected using auxiliary register. For example, if one of the value fields is to be replaced by another value, the replacing value is first assembled in the auxiliary register and it is routed into the write head instead of the buffer register. To delete an EWCR, the buffer register is "shorted" and the incoming tuple is written directly onto the track. After fields are evaluated while exiting, the output of the buffer unit is connected to the I/O unit. The organization of the buffer unit is shown in Fig. 5.12 for both options. Note, in option I, the buffer is divided into two parts because the EWCR's are of different size in two halves of the track. While writing onto the track, the incoming EWCR record provides the synchronization. Note, all the records are of the same length. For the last record, the TEM provides the synchronization. For this purpose, the TEM is chosen in such a way that its length and format is similar to a record except for the value field, which contains a special pattern.

#### 5.5.4 Search Unit (SU)

This unit evaluates the condition or the qualification on the fields of the EWCR's. Firstly, it evaluates the specification of constituent name(s) indicated in the WCRML instructions. Then it performs comparisons on the value fields as specified by the instructions. The operands are supplied by the control unit prior to starting of comparison. The overall organization of search unit



Option I (Extra Hardware)

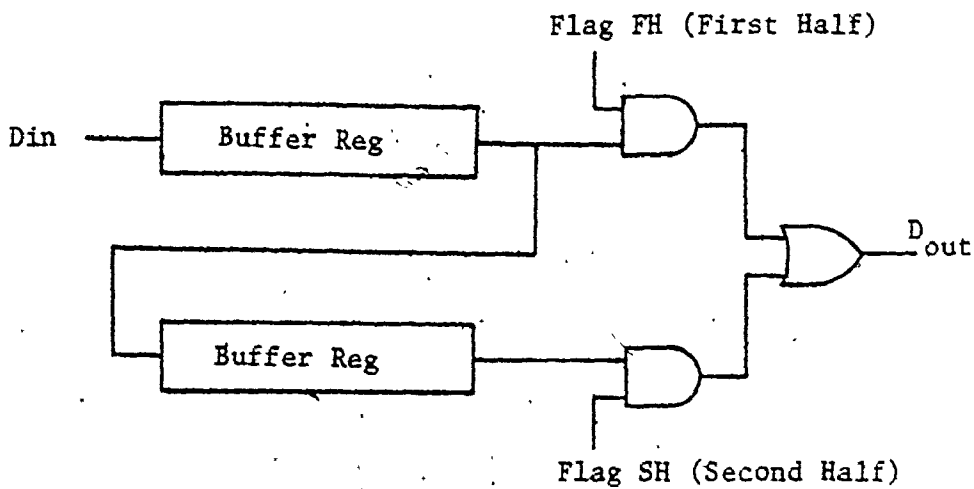


Fig. 5.12 Buffer Unit



is shown in Fig. 5.13 a. Each comparison unit consists of a shift register which holds the comparand and the associated logic for performing bit serial comparison. The comparison operator for each of these units is supplied by the control unit. The final result match/mismatch is asserted by an indicator flip flop. These details are shown for one such unit in Fig. 5.13 b.

As mentioned earlier, the qualifications within a cell are either in conjunction or disjunction. This is performed by the gating circuit and the final result is asserted. This signal enables operations to be done on the qualified items.

#### 5.5.5 Arithmetic and Logic Unit (ALU)

Each cell has an ALU which performs set function computations and arithmetic updates, such as MIN, MAX, ADD, SUB, etc. These operations are performed within the cells avoiding any transporting of data out of cells. The overall results in the case of set function operations are computed by the QP. The required hardware is shown in Fig. 5.14. Note, the arithmetic operations have to be performed within the time delay provided by the buffer register. For this reason, a local clock which has a higher clock rate than the bit rate is used. All arithmetic is performed serially using standard circuits. For set functions the following scheme is adopted. MIN/MAX is obtained by replacing the present extremum value in a register with the incoming value only if it qualifies to be the next extremum. SUM is obtained by cumulative addition of the incoming items. CAR (cardin-



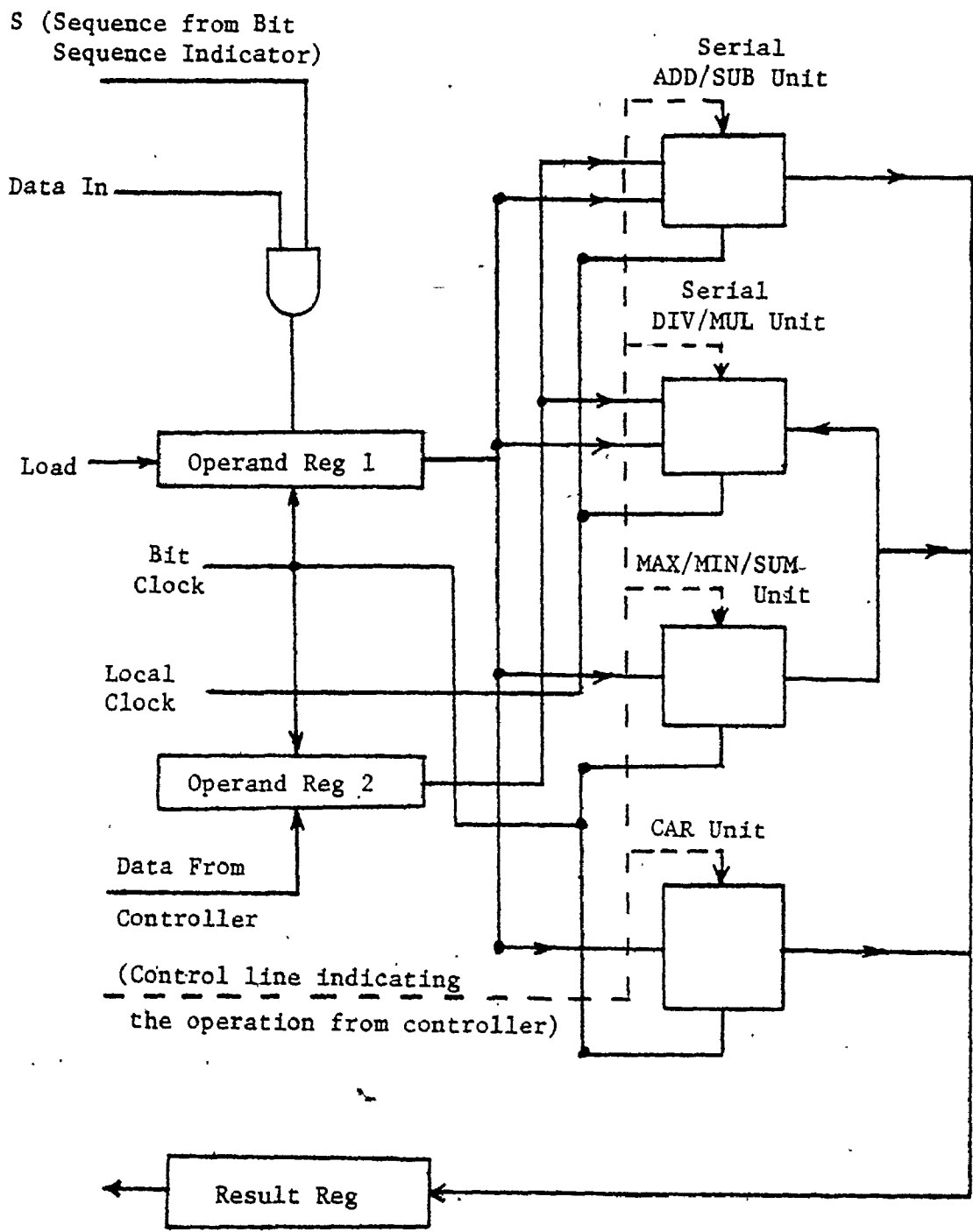


Fig. 5.14 ALU Unit

ality) is applicable for only key values. Since key values are distinct, CAR is obtained by either counting the incoming items or marking RAM bits and counting the marked RAM bits. Note, in most queries, CAR is associated with the key values only. (For example, 'how many employees earn more than 20k', 'how many salesmen live in Toronto', etc.). In option I, this scheme would require more refinement due to key duplication.

#### 5.5.6 RAM Logic Unit (RLU)

Each cell consists of a one bit RAM whose length is equal to the number of key value fields on the track. The RAM and the associated logic is shown in Fig. 5.15. In option I the address register inputs are connected to either the outputs of a counter, which counts the EWCR's on the track, or a serial in parallel out register, which would be loaded with the address field from the incoming data (at the appropriate time). In option II, the address register is connected to the counter only. The counter may be clocked by bit sequence indicator corresponding to EWCR's. For allowing transfer of RAM contents between a cell and the QP, a faster local clock is used by the QP. Thus, a cell RAM is directly accessible by the QP, if it is not being used by the cell. The output of the RAM is also used for qualification evaluation.

#### 5.5.7 I/O Unit (IOU)

The I/O unit consists of a set of registers and a buffer

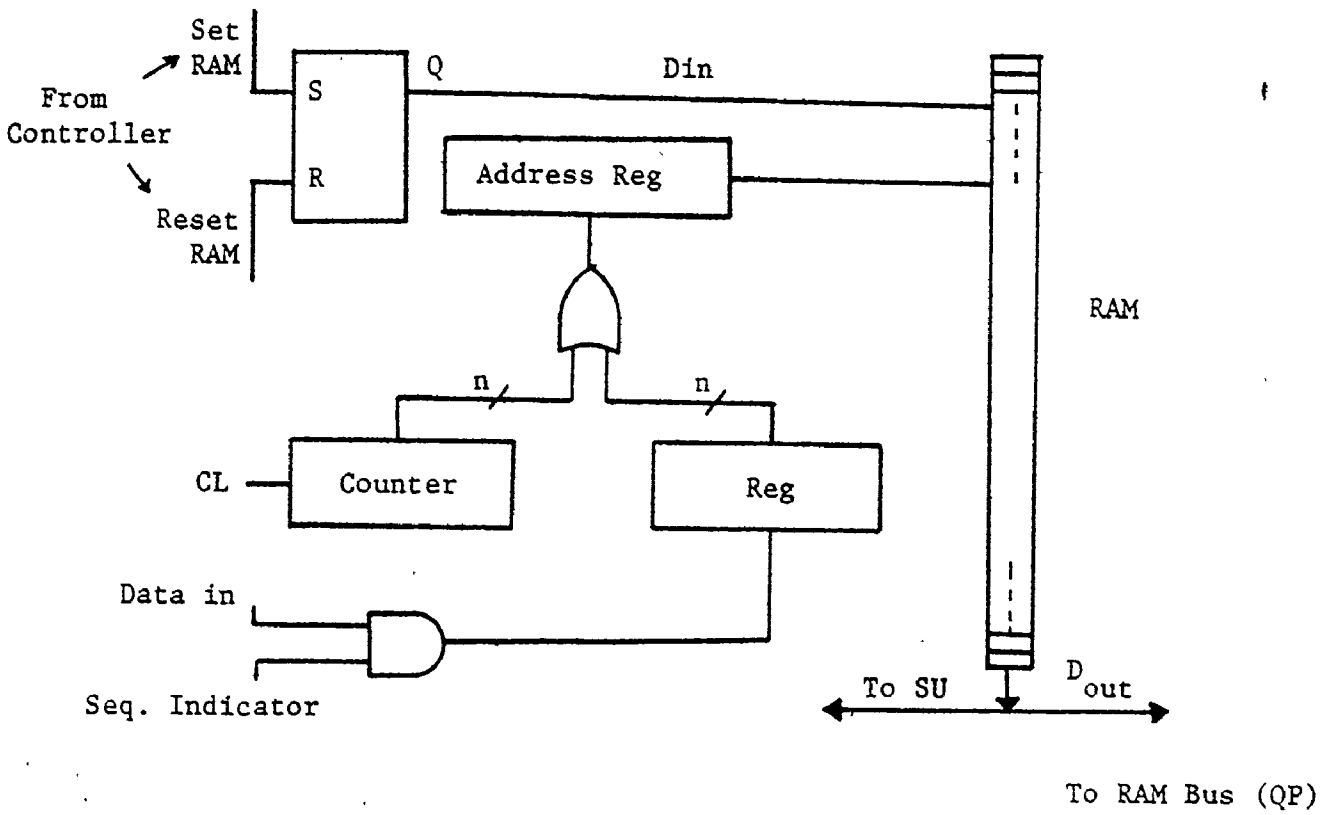
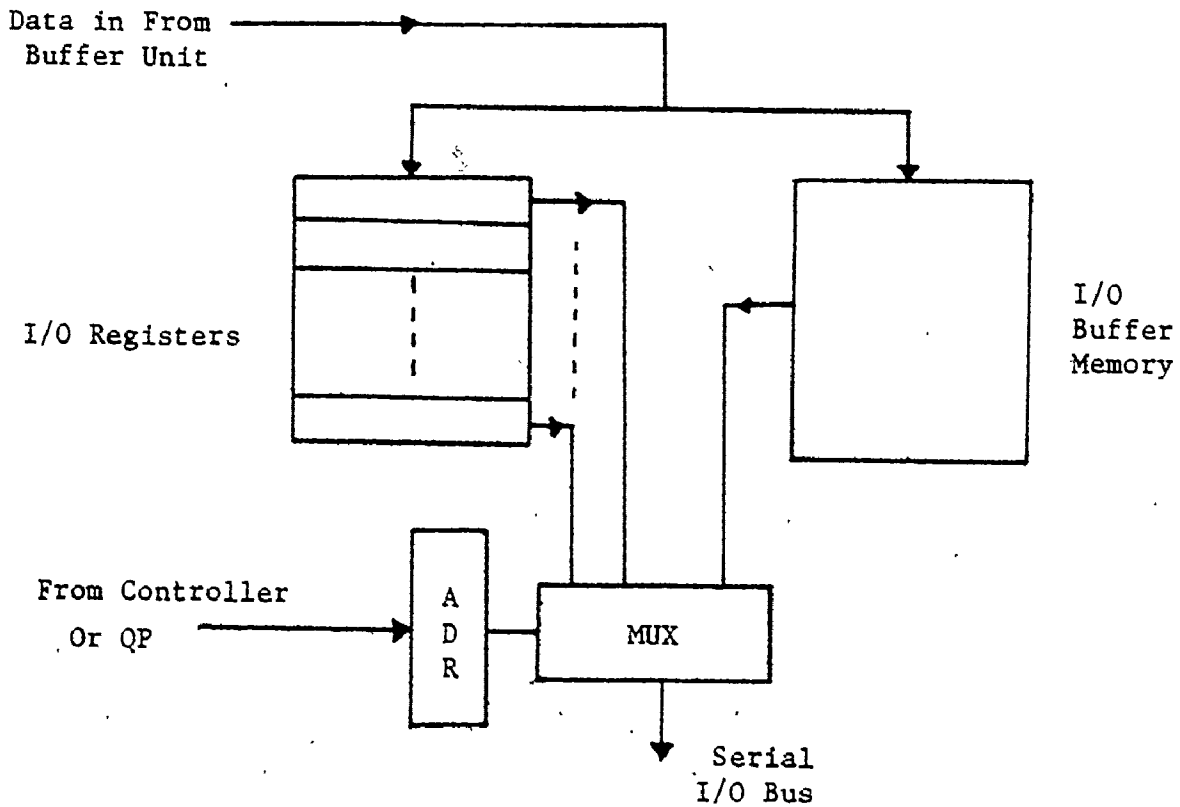


Fig. 5..5 RAM Logic Unit



memory. The outputs of these, are multiplexed onto a serial I/O bus which is accessible by the QP. The required register or buffer may be selected by placing a control word into the multiplexer. The I/O unit is accessible by the QP, if it is not being used by the control unit of the cell. The input to the IOU is from the output of buffer register in Buffer unit. Fig. 5.16 shows the organization of this unit. The registers can hold only one value at a time, while the buffer can hold several qualified values. The buffer is used for JOIN operation by the QP while the registers are used for TRANS\_REG operation.

#### 5.5.8 Control Unit (CU)

The control unit is responsible for overall co-ordination of the various functional units mentioned above. In addition, it also receives the WCRML instruction set and the associated data from the QP and executes the instructions in the cell. It receives the various control signals such as track identification indicators, qualification results from all the units and issues the proper control signals for further operations at the appropriate time. The overall organization of the control unit is shown in Fig. 5.17. There are two ways of implementing the control unit; a) using a microprocessor, b) using a microprogrammed control unit. If a microprocessor ( $\mu$ p) such as Intel 8085 is used, the response time of a cell would be very slow due to the cycle time of the  $\mu$ p. However, this would provide an added advantage of eliminating the ALU unit, because the  $\mu$ p can perform these

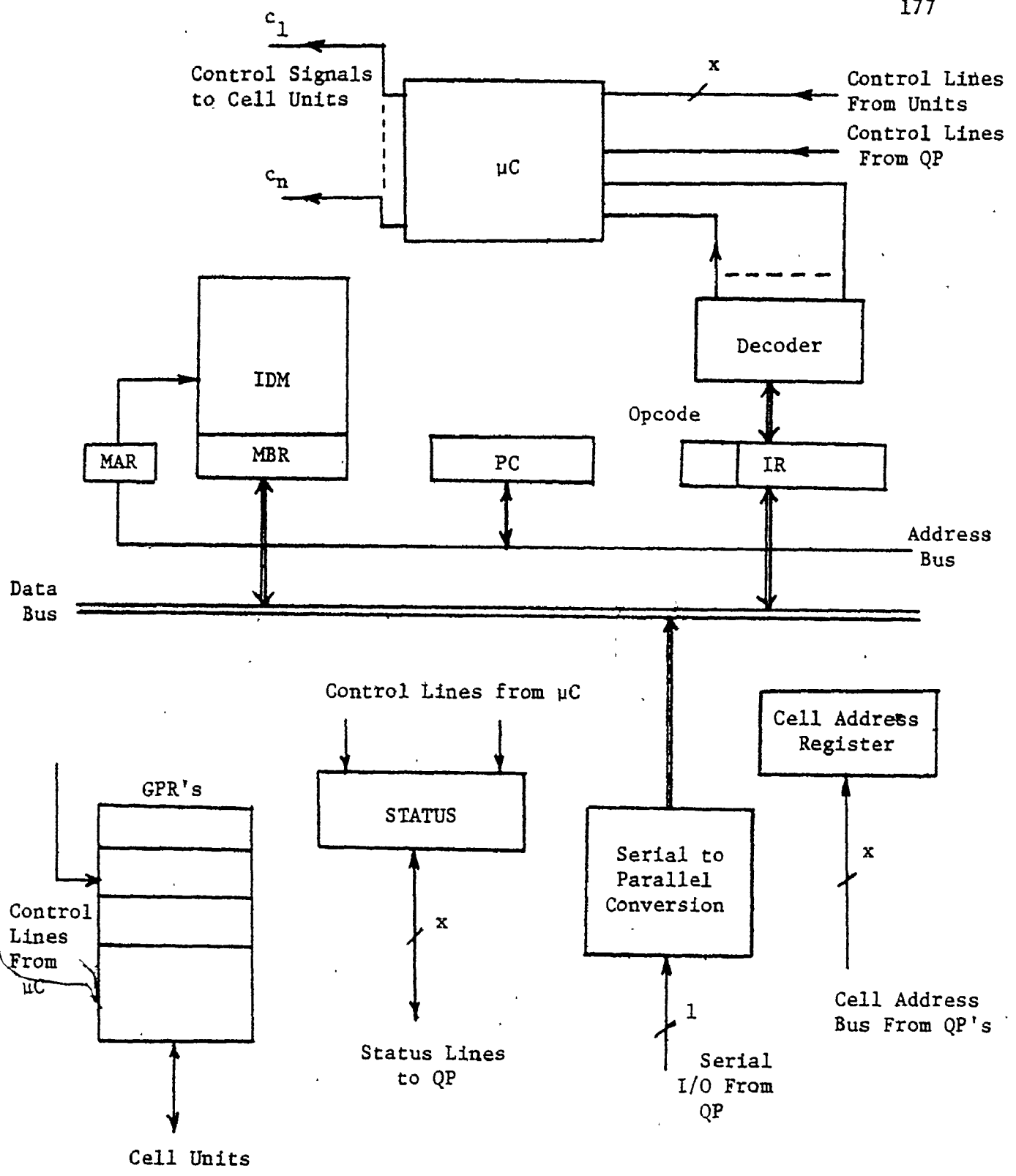


Fig. 5.17 Control Unit

operations. Also, since the word length of commercially available  $\mu$ p's is small when compared to the value items on the track, the software overhead would lead to slower response time. On the other hand, microprogrammed control unit would provide a faster response due to complete hardware design. Fig. 5.17 shows the  $\mu$ programmed version of the control unit.

The control unit consists of the following major units:

- a) Instruction and Data Memory (IDM)
- b) Memory Address Register (MAR)
- c) Memory Buffer Register (MBR)
- d) Program Controller (PC)
- e) Instruction Register (IR)
- f)  $\mu$ programmed Controller ( $\mu$ C)

The QP processor loads the IDM with the machine encoded WCRML instructions and the data. Once control unit is initiated by the QP, it fetches the instructions and the data from the IDM into IR. Then, it transfers the data to SU and ALU (depending upon the instruction). It decodes the instruction and provides the decoded information to the  $\mu$ c. The  $\mu$ c utilizes this instruction information and the other control lines from the various functional units and provides the required sequence of control signals to all the units.

The control unit has a set of general purpose registers used for program control as well as for storing some intermediate



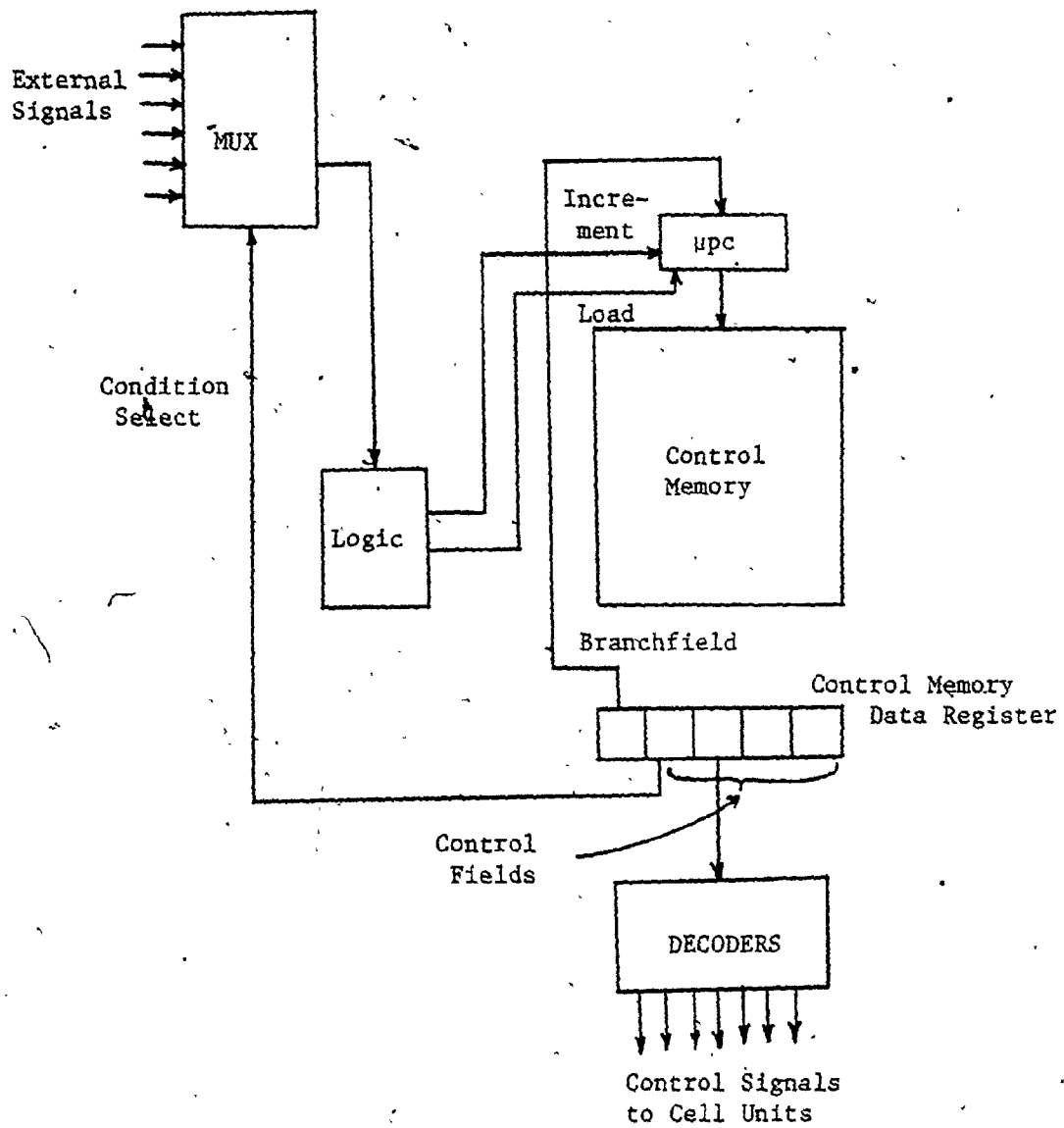


Fig. 5.18  $\mu$ programmed Controller

values, such as SUM, MIN, etc. It has two more registers called Status Register and Cell Address Register. These are used for communication between the QP's and the cells. The status register consists of status bits such as active cell, empty cell, I/O active, etc. To poll a cell, the QP simply broadcasts the cell address one after another. The polled cell would enable its status on to the status bus. The QP receives the status and checks for the required bit. For example, if the cell is inactive, the QP would send a control signal to the  $\mu\text{C}$  of the polled cell. the  $\mu\text{C}$  in turn, sets a status bit indicating it is active and no other QP can enslave the cell.

The  $\mu$ programmed controller ( $\mu\text{C}$ ) would be typically as shown in Fig. 5.18. It consists of a control memory (CM), a micro program counter ( $\mu\text{PC}$ ) and a control memory data register (CMDR). The sequences of operations are converted into a  $\mu$ program code and loaded into the CM. The  $\mu$  instructions are fetched into CMDR and decoded to give the required signals. The  $\mu$  instructions select the external conditions to be checked and if they are satisfied, the program branches to the location specified by the branch field. The detailed chip level design and the  $\mu$ programmed instructions are out of the scope of this thesis. In the next two sections, the WCRML instruction times and the hardware of a QP are outlined.

#### 5.6 Cell Executed Instruction Times

The retrieval instructions in both options are executable

in one revolution of the cell. No additional hardware is required for this purpose. Once the operands are supplied to the cell, the qualification is evaluated and the marking, resetting or reading of the marked values is performed. In option I, the marking of second constituent values which go with the marked values of the first constituent values is accomplished using the RAM. However, if marking of both second constituent values and the RAM is required, it would take  $1\frac{1}{2}$  revolutions in option I (for SET instruction). All the set function instructions require only one revolution time. No additional hardware is required in option I.

Other instructions such as CREATE, DESTROY need only one revolution in both the options. For creating a new cell, the format of the header and an EWCR are supplied by the QP to the cell - The first two (three for option I) general purpose registers of the controller. The update instructions, ADD, SUB, MUL and DIV can be executed in one revolution in both the options. However, the execution of INSERT, DELETE and MODIFY differs significantly in option I and II. In option II, they would require only 1 revolution because the tuples are unordered and both the constituent values are together. In the case of option I, these instructions are executed as described below.

To insert a key value - nonkey value pair in a cell with empty space (indicated by the status) of option I, the following sequence of operations have to be performed by the control unit:

- 1) Find the EWCR in the first half of the track with a key value

greater than the value to be inserted. Insert an EWCR with the key value in that position using the auxiliary register. Note the address of the inserted EWCR ( $A_1$ ).

- ii) Move the following EWCR's down the track using the buffer and the auxiliary registers.
- iii) Insert the nonkey value of the end of the second half of the track as an EWCR with the value and address field  $A_1$ , as in step i). Note the address of the newly inserted EWCR ( $A_2$ ).
- iv) Insert  $A_2$  into the address field of key value inserted in step i).

Clearly, these operations take  $1\frac{1}{2}$  revolutions. However, if the track is full, it would take more than  $1\frac{1}{2}$  revolutions because the overflow would have to be inserted into the next track. The maximum number of revolutions would be  $1\frac{1}{2} * (\# \text{ of tracks which are full and contain values greater than the value to be inserted})$ . For DELETE instruction, the values are marked for deletion only if no address fields point to this value. Otherwise, only the corresponding address field pointers of the other constituent are changed to null. In case of an EWCR deletion, the sequences of operations are similar to insertion except that it is deleted instead of being inserted. The number of revolutions is still the same ( $1\frac{1}{2}$  revolution). MODIFY is executed by first deleting the values and then inserting the new values. The instruction times are summarized below.

Instruction	Time in Revolutions	
	Option I	Option II
SET	1	1
RESET	1	1
READ	1	1
MIN	1	1
MAX	1	1
SUM	1	1
CAR	1	1
AVG	2	2
INSERT	$1\frac{1}{2} * n$	1
DELETE	$1\frac{1}{2} * n$	1
MODIFY	$3 * n$	1
ADD	1	1
SUB	1	1
MUL	1	1
DIV	1	1
LOAD	negligible	negligible
CREATE	1	1
DESTROY	1	1

n = number of tracks that are full following the track into which the value is to be updated

## 5.7 Query Processor Hardware

Each query processor is implemented on a micro/minicomputer with some additional hardware. The overall organization is shown in Fig. 5.19. The QP receives the WCRML programs from the conceptual processor and broadcasts them to selected cells. It also computes the overall set function results and co-ordinates the cells for inter-cell communication.

The functional units of a QP are the following:

- i) Boolean Evaluator (BE)
- ii) Join and Sort Unit (JSU)
- iii) RAM POOL
- iv) Controller ( $\mu$ processor/minicomputer)

### 5.7.1 Boolean Evaluator

As mentioned earlier, a conditional expression in WCRL may involve AND/OR combination of conditional clauses on the attributes. These individual conditional clauses are evaluated in the cells while the overall evaluation of conditional expression is done by the Boolean evaluator. This instruction BOP\_MARK is used for this purpose. This instruction evaluates the overall condition on the RAM pools of PCP's.

Any general boolean expression can always be expressed as sum products or product of sums [MILLER 65]. Let  $n$  be the number of input variables and  $m$  be the number of terms in the sum of product form.

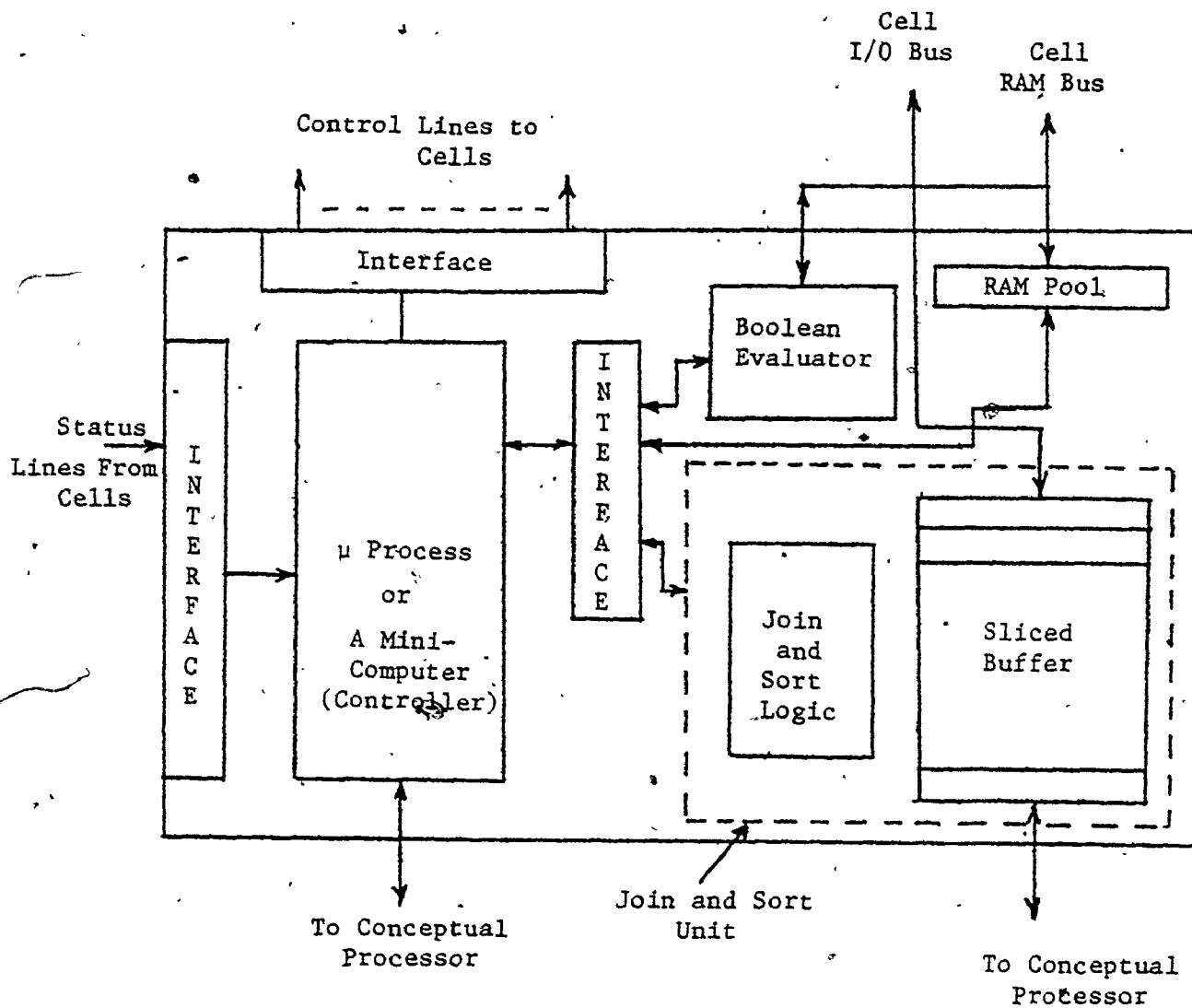


Fig. 5.19 Organization of a QP

i.e.

$$f = p_1 + p_2 \dots + p_m \quad (1)$$

where

$$p_i = q_1 \cdot q_2 \dots q_n$$

and

$$q_i = q_i \text{ or } q_i'$$

A boolean expression such as  $f$  is evaluated as follows.

For each product term  $p_i$ , a vector  $s_i (a_{i1}, a_{i2}, \dots, a_{in})$ , where

$$a_{ij} = \begin{cases} 0 & \text{if } q_j \text{ is involved in } p_i \\ 1 & \text{if } q_j \text{ is not involved in } p_i \end{cases}$$

These vectors are stored in a random access memory as shown in Fig.

5.20. Now, the product terms is evaluated sequentially using  $n$  - number of R-circuits. An R-circuit allows the corresponding boolean variable  $q_i$  to pass through only if  $a_{ij} = 0$ ; otherwise, the output of R-circuit would be 1. The sum of the products is achieved by an OR gate. After  $m$  clock pulses, the boolean function would have been evaluated and the result would be available at  $f$  (Fig. 5.20). The memory is addressed by a counter, which gets reset after  $m$  clock pulses. Now, the next set of inputs  $q_1$  through  $q_n$  may be evaluated following the same procedure.

The time taken to evaluate  $f$  would be  $(m \cdot t_a)$  where  $t_a$  is the access time of the memory. Therefore, it would be of the order of  $\mu$  seconds to evaluate a reasonable boolean expression  $f$ . The  $A_i$  vectors can be obtained from the conditional expression during query evaluation



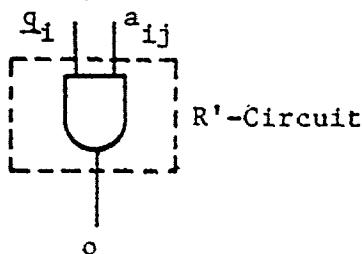
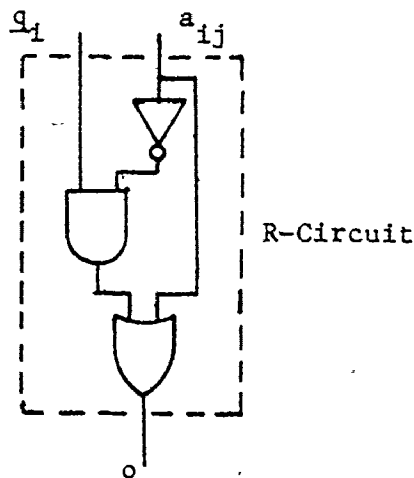
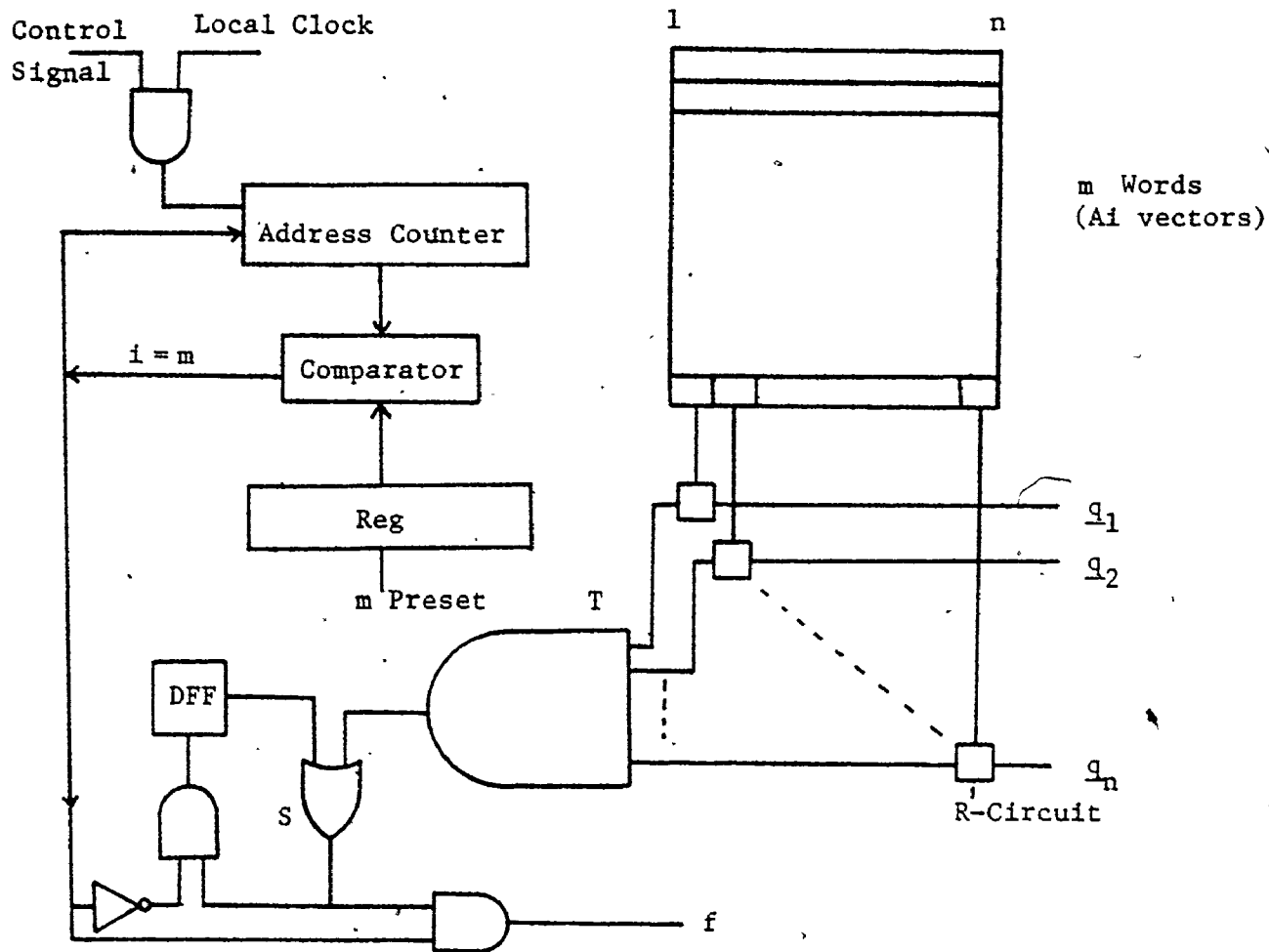


Fig. 5.20 Boolean Evaluator

by the conceptual processor and supplied to the QP.

Since there always exists a product of sum equivalent for any expression  $f$ , the evaluation can be alternatively carried out using a slightly modified hardware. The changes to be made to Fig. 5.20 are the following:

- i) Replace T by an n-input OR gate
- ii) Replace S by an AND gate
- iii) Replace R by R' circuit (AND gate)

It may be noticed that, in this case the number of gates required to implement the hardware is much less than the previous scheme.

#### 5.7.2 Join and Sort Unit

Join is one of the most important functions performed by any data base machines. So far, the trend in designing hardware to implement the join has been restricted to implicit join only. While RAP [OZKARAHAN 75] and CASSM [SU 79] implement only implicit join, LEECH [McGREGOR 76] and CAFS [BABB 79] implement full joins by selecting the rows and sending them to a general purpose processor. None of the machines provide a hardware-supported join facility. In this section, a hardware unit, using associative memory, is proposed. This unit is also capable of performing partial sorting.

As mentioned earlier in WCRC, the JOIN instruction is executed by a QP. It allows for both natural join and cross product.

Natural join is sufficient if the conditional expressions do not involve inter-attribute clauses. If these clauses are present, cross product of the EWCR's is inevitable. At first, the hardware to perform natural join over a key domain is presented. This is slightly modified to accommodate cross product.

The overall organization of this unit is shown in Fig. 5.21. It consists of an associative memory (AM) and a buffer memory (BM). The associative memory is used for associating a unique address with each key value - nonkey value pair read from the I/O buffer memories of the cells. Consider an example where several PCP's with a common key have to be joined on the marked key values of one of the PCP's. The algorithm to perform such a join is summarized as follows.

Algorithm 1 (Natural Join on Keys)

- i) Read the marked key and nonkey value pairs into the data part of the associative memory from the PCP which contains the marked key values. If sorting is required on a particular attribute, transfer these marked key values to the PCP containing this attribute and read this PCP instead, into the associative memory.
- ii) Sort the associative memory on the nonkey domain and set up the address part of the associative memory such that the minimum valued data item has the starting address of the buffer; i.e. the addresses indicate the sorting order of the values.
- iii) Read the next PCP and for every value pair, use the key value as the comparand to AM to obtain the address associated in the out-

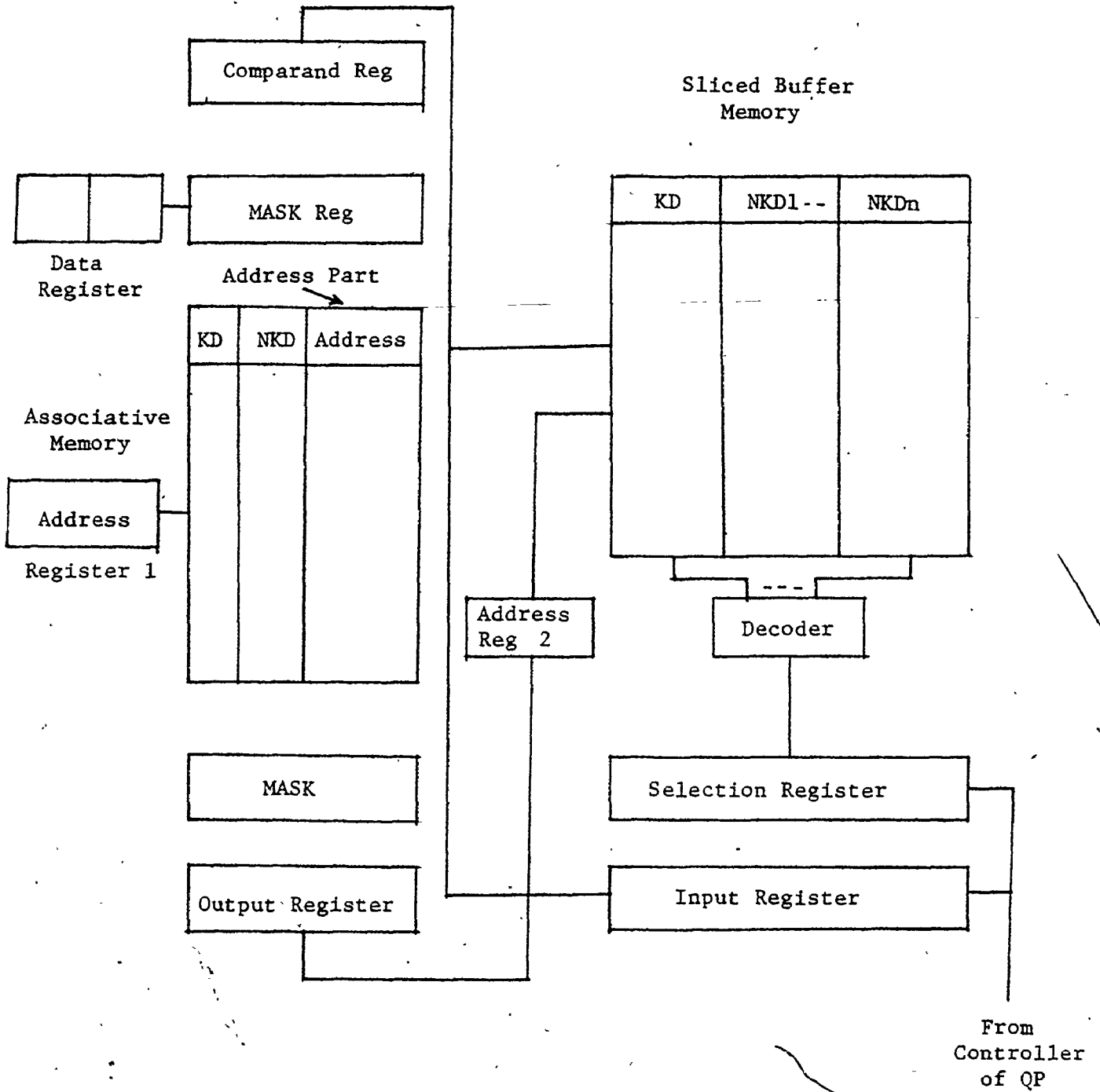


Fig. 5.21 Join and Sort Unit

put register. Use this address to store the nonkey value into the appropriate buffer slice.

iv) Repeat step iii) for all the PCP's involved in the join. At the end of these operations, the sliced buffer will have the required tuples formed by joining the PCP's over the key.

The selection of the slices of the buffer can be controlled using a selection register and a decoder (Fig. 5.21). The selection register gets its information from the QP controller. To accomplish the above operations, the associative memory must have the following features:

- a) Bit parallel word serial write
- b) bit parallel word serial read
- c) word parallel bit serial comparison

The design of one such memory is shown in Fig. 5.22. It consists of a data part of M words, each N bit long and an address part, M words of k bits each. The data and commands are issued by the controller to every cell in the memory, which is capable for performing read, write or search.

The comparand register is used to hold the key value. The mark register is used to choose selected bits of the comparand register for search. Associated with each word, there is a response store, which indicates match/mismatch. A separate 'SET' line is used by controller to set all the response stores at the beginning.

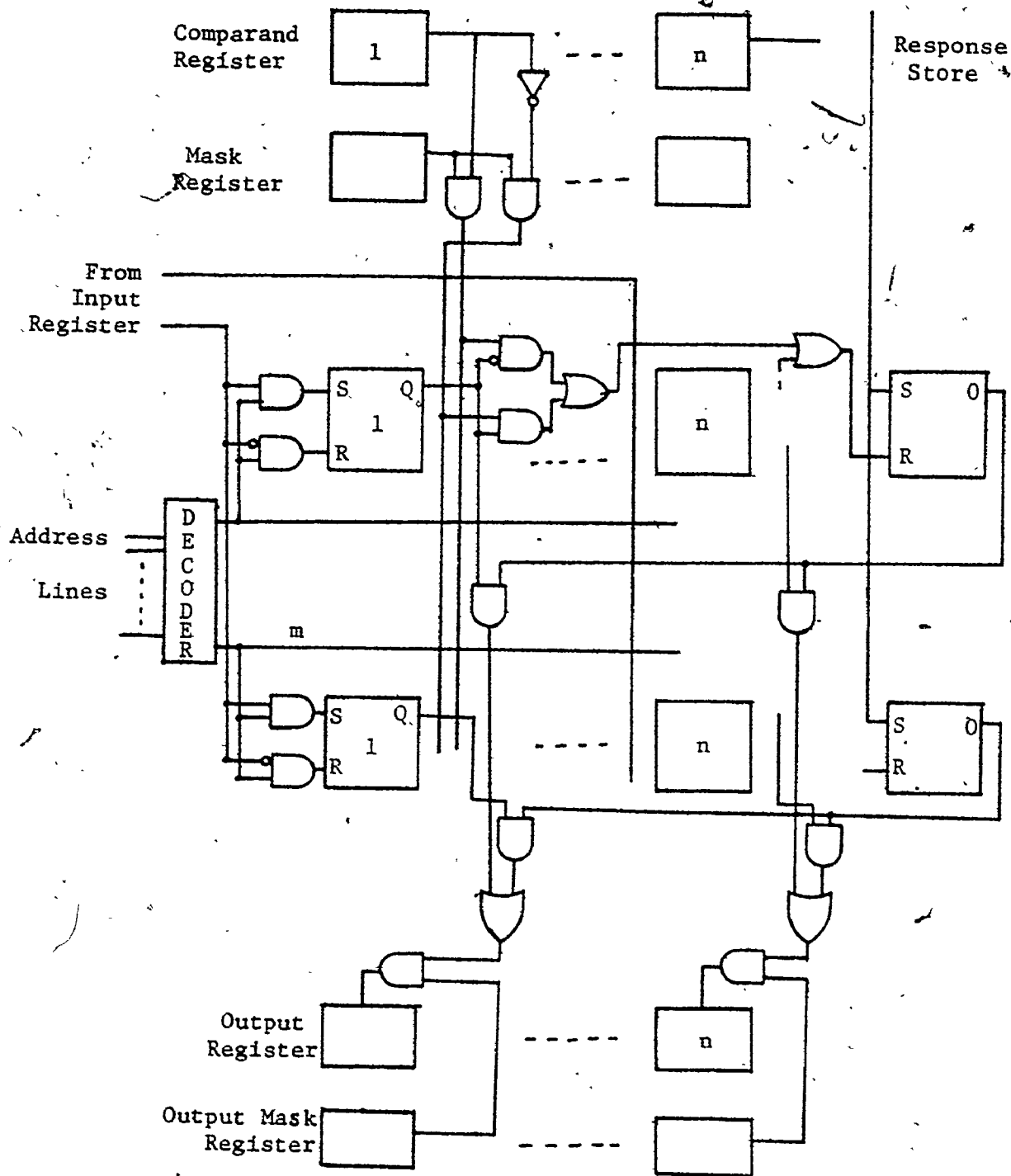


Fig. 5.22 Associative Memory (Data Part)

of a search. After the search, only the word with matched value will have its response set, the rest will be reset. This word would be available in the output register. The address part of the associative memory does not have search logic associated with the cells because only data part is searched or sorted.

The algorithm for sorting the nonkey domain of the data part is described below.

Algorithm 2 (SORT)

- i) Find the first available minimum and maximum of the nonkey domain of the words in one interrogation of the associative memory.
- ii) Set up the address part of the minimum word to contain the starting address of the buffer and that of the maximum to have the last address of the buffer.
- iii) Set up the indicator flags of the words processed.
- iv) Repeat step i) through iii) until all the indicators are set.

In step i) of the algorithm, the maximum and minimum are obtained into the comparand and output registers, respectively, by the hardware algorithm shown in Fig. 5.23. The extra hardware needed to implement this algorithm is an additional flip flop with each word (indicator flag), a counter to control the hardware program and a few gates to find out if there are 'some' (or 'all') responders. To sort  $M$  data items of  $P$  bits wide, it would take  $(P \cdot t_{cl} + M \cdot t_{RT})$  time, where  $t_{cl}$  is the clock rate and  $t_{RT}$  is

C - Comparand Register  
 O - Output Register  
 M - Mask Register

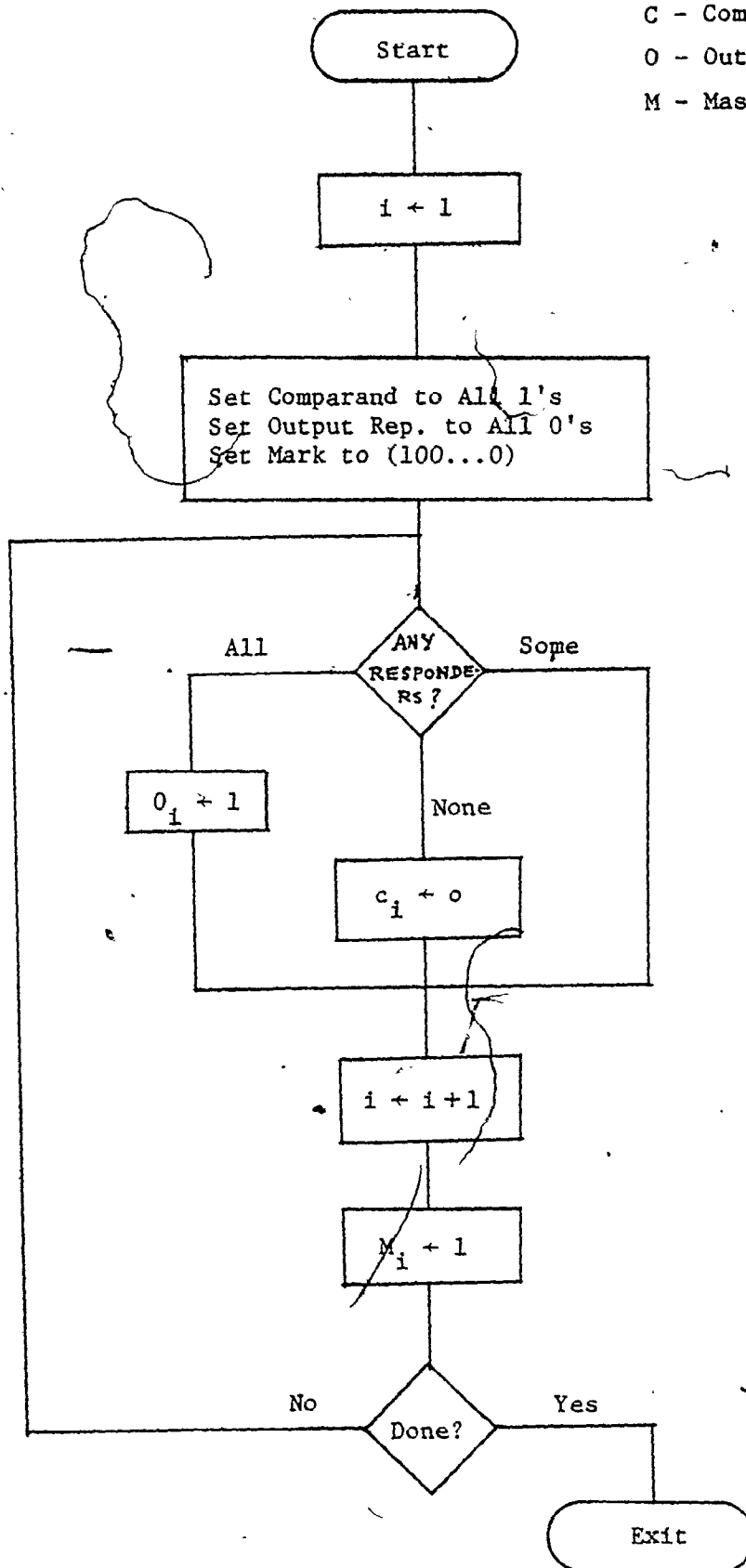


Fig. 5.23 Flow Chart to Find Maximum and Minimum



the time taken for a register transfer. Note, this is of the order of  $\mu$  seconds because this involves reading the marked values from the cell I/O buffers, loading and interrogating the associative memory and loading of the buffer memory. Thus, the overall time for join would be well within the order of one revolution time of a cell ( $t_r$  is about .2 to .5 seconds typically).

#### Theta Join on Nonkey Values

As mentioned before, if the conditional expression involves inter-attribute clauses, the natural join may not always suffice. In this case, the EWCR's have to be taken one at a time and a natural join or cartesian product may have to be performed to obtain the tuples. For example, consider two PCP's  $P_1[K_1; A]$  and  $P_1[K_1; B]$  which have to be joined only if  $A \geq B$  is satisfied. This can be done as follows.

- 1) Take one EWCR from  $[K_1; A]$  and pass A value to the PCP  $[K_1; B]$ . Mark the B values that satisfy the condition and mark  $K_1$  values which go with these. Join (natural) the two PCP's over the  $K_1$  values of  $P_1$ .
- 2) Step 1 is repeated for all marked EWCR's in the PCP  $P_1$ .

If the second PCP is of the form  $P_2[K_2; B]$ , then a cross product has to be performed in step 1. In a more general case, where we have more than two PCP's, the cross product has to be performed on  $n$  PCP's. This can be done as follows.

Let  $n_1$  be the number of values in the first PCP,  $n_2$  be the number of values in the second and  $n_n$  be the number in the  $n$ th PCP. Obviously, the cartesian product will have  $n_1 * n_2 * n_3 * \dots * n_n$  ( $n$ ) tuples. Each value in the first PCP is repeated in these tuples  $n_2 * n_3 * \dots * n_n$  times. Once all values are repeated this many number of times, the whole operation is repeated  $\frac{n}{n_1 * n_2 * n_3 * \dots * n_n}$  times. Similarly, for the second PCP, each value is repeated  $n_3 * \dots * n_n$  times and the whole operation is repeated  $\frac{n}{n_2 * n_3 * \dots * n_n}$  times. For the  $i$ th PCP, each value is repeated  $n_{i+1} * \dots * n_n$  times and the operation is repeated  $\frac{n}{n_i * \dots * n_n}$ . An example illustrating this procedure is shown in Fig. 5.24. Note, for the  $n$ th PCP, the values are repeated only once.

From the above procedure, if  $n_i$ 's are known beforehand, the cartesian product can be formed in the sliced buffer with the PCP's read in parallel or serial. The extra hardware required is an additional buffer to hold the marked values from one PCP and a counter to obtain  $n_i$ . If sorting is required on any attribute, the values of the corresponding PCP are first read into the additional buffer and sorted using the AM, as described before, and the corresponding bit slice is filled in.

### 5.7.3 RAM Pool

The QP consists of a RAM pool of size equal to the maximum size of PCP RAM pools in the system. This is used for transferring RAM contents between cells/PCP's.

<u>A</u>	<u>B</u>	<u>C</u>	<u>D</u>
$a_1$	$b_1$	$c_1$	$d_1$
	$b_2$	$c_2$	$d_2$
		$c_3$	
$n_1 = 1$	$n_2 = 2$	$n_3 = 3$	$n_4 = 2$

Cross Product

$a_1$	$b_1$	$c_1$	$d_1$
$a_1$	$b_1$	$c_1$	$d_2$
$a_1$	$b_1$	$c_2$	$d_1$
$a_1$	$b_1$	$c_2$	$d_2$
$a_1$	$b_1$	$c_3$	$d_1$
$a_1$	$b_1$	$c_3$	$d_2$
$a_1$	$b_2$	$c_1$	$d_1$
$a_1$	$b_2$	$c_1$	$d_2$
$a_1$	$b_2$	$c_2$	$d_1$
$a_1$	$b_2$	$c_2$	$d_2$
$a_1$	$b_2$	$c_3$	$d_1$
$a_1$	$b_2$	$c_3$	$d_2$

Figure 5.24 Cartesian Product

#### 5.7.4 Controller

The controller is responsible for co-ordinating the various units in the QP. It is also responsible for computing the overall set function results. Since QP is implemented in a  $\mu$ p/minicomputer, no additional hardware is required for this purpose. The controller receives WCRML instructions and the parameter data from the conceptual processor and enslaves the required cells using the status bus. Once the resultant tuples are obtained in the join unit, the controller sends these tuples to the conceptual processor. It also checks the cell status for any error messages and conveys them to the conceptual processor. Of all the instructions executed by QP, only JOIN takes considerable time, the execution times for other instructions is negligible compared to revolution time of a cell.

#### 5.8 Summary

In this chapter, the details of storage structures, WCRML and the hardware organization of cells are reported to a moderate level of complexity. The outlines of cell control unit and the QP hardware are discussed. In the next chapter, the performance of WCRC is evaluated and compared with some other schemes.

## CHAPTER VI

### PERFORMANCE EVALUATION

#### 6.1 Introduction

In this chapter, the performance of WCRC is compared with that of GDBMS [DOGAC 80] and DeFiore's associative scheme [DEFIORE 74]. Note the comparison will be based on the storage structures only. For a class of data base systems based on hierarchical structure of relations, general expressions are derived for storage in number of bits, and retrieval and update times in number of interrogations.

#### 6.2 Storage Requirements

The storage requirements for a hierarchy of relations, when data is stored as PCP's is compared with the same when stored as large n-ary relations as in GDBMS or DeFiore's associative scheme.

The following assumptions are made while comparing the storage requirements of the three schemes:

- i) In all schemes, the storage needed for track headers, gaps, mark bits, etc. are not taken into consideration. Only the storage needed for domain values and address fields are considered.
- ii) In all schemes, keys are stored in coded (binary) format.

Consider the hierarchical structure with n-level embedding as shown in Fig. 6.1. Each level  $i$  corresponds to a relation  $R_i$ . Each  $R_i$  has  $m_i$  simple domains  $\underline{a}_{i1}$  to  $\underline{a}_{im_i}$ ; and a non-simple domain  $R_{i+1}$ , which in turn is a relation of  $m_{i+1}$  domains at the next level  $i+1$ .

The associative normal form (ANF) for this structure as described by [DEFIORE 74] is as follows:

$$\begin{aligned}
 R_1 & (\underline{a}_{11}, \underline{a}_{12}, \dots, \underline{a}_{1m_1}, \underline{\alpha}_1) \\
 R_2 & (\underline{a}_{21}, \underline{a}_{22}, \dots, \underline{a}_{2m_2}, \underline{\alpha}_1, \underline{\alpha}_2) \\
 R_3 & (\underline{a}_{31}, \underline{a}_{32}, \dots, \underline{a}_{3m_3}, \underline{\alpha}_1, \underline{\alpha}_2, \underline{\alpha}_3) \\
 & \vdots \\
 & \vdots \\
 R_n & (\underline{a}_{n1}, \underline{a}_{n2}, \dots, \underline{a}_{nm_n}, \underline{\alpha}_1, \underline{\alpha}_2, \dots, \underline{\alpha}_{n-1})
 \end{aligned}$$

Here,  $\underline{a}_{ij}$ 's denote the simple domains.

$\underline{\alpha}_i$ 's denote simple domains to link the relations from one level to another, i.e., they are the pointers.

Note,  $\underline{\alpha}_i$  at level  $i$ , is nothing but the key domain of the relation,  $R_i$ , because every tuple in  $R_i$  is associated with a unique value of  $\underline{\alpha}_i$  domain. The ANF form would be stored in associative memory as shown in Fig. 6.2 In GDBMS, they would be stored as n-ary relations.

The storage requirement for associative scheme,  $S_1$ , as

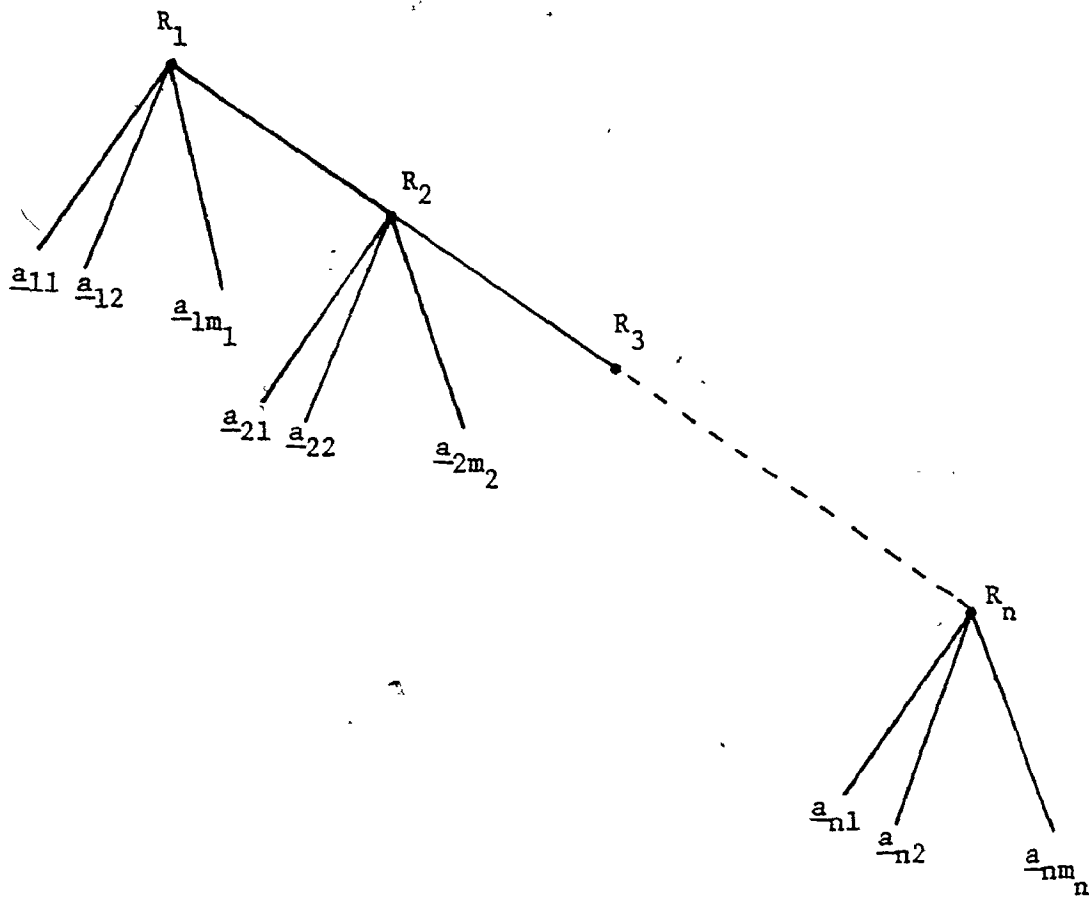


Fig. 6.1 A Hierarchy of Relations with  $n$ -Level Embedding

Response Store

$l_1$

$l_n$

$R_1$	$a_{11}$	$a_{12}$	-----			$\alpha_1$
$R_n$	$a_{n1}$	$a_{n2}$	-----	$\alpha_{n-1}$		$\alpha_1$

Fig. 6.2 ANF in Associative Memory



shown by DeFiore is given by the following expression,

$$S_1 = \sum_{i=1}^n \sum_{j=1}^{m_i} a_{ij} \ell_i + \sum_{i=1}^n R_i \ell_i + \begin{cases} \sum_{i=1}^n \ell_i \sum_{k=1}^i \alpha_k - \ell_n \alpha_n & n > 1 \\ 0 & n=1 \end{cases} \quad (6.1)$$

where,

$\ell_i$  - represents the number of tuples in relation  $R_i$   
(length of  $R_i$ )

$n$  - denotes the level of embedding

$a_{ij}$  - represents the storage needed for storing the domain value  $a_{ij}$  (in bits)

$R_i$  - denotes the storage in bits for storing relation number  $i$

$\alpha_i$  - denotes the storage for one value of domain  $\alpha_i$

By expanding and regrouping the terms, Eq. 6.1 can be simplified as follows

$$S_1 = \sum_{i=1}^n \sum_{j=1}^{m_i} a_{ij} \ell_i + \sum_{i=1}^{n-1} \alpha_i \ell_i + \sum_{i=1}^{n-1} \alpha_i \sum_{k=i+1}^n \ell_i + \sum_{i=1}^n R_i \ell_i \quad (6.2)$$

In GDBMS, the hierarchy is stored as n-ary relations, except that the relation number is not repeated with each tuple. The storage ( $S_2$ ) required would be same as Eq. 6.2 except the last term.

$$S_2 = \sum_{i=1}^n \sum_{j=1}^{m_i} a_{ij} \ell_i + \sum_{i=1}^{n-1} \alpha_i \ell_i + \sum_{i=1}^{n-1} \alpha_i \sum_{k=i+1}^n \ell_i \quad (6.3)$$

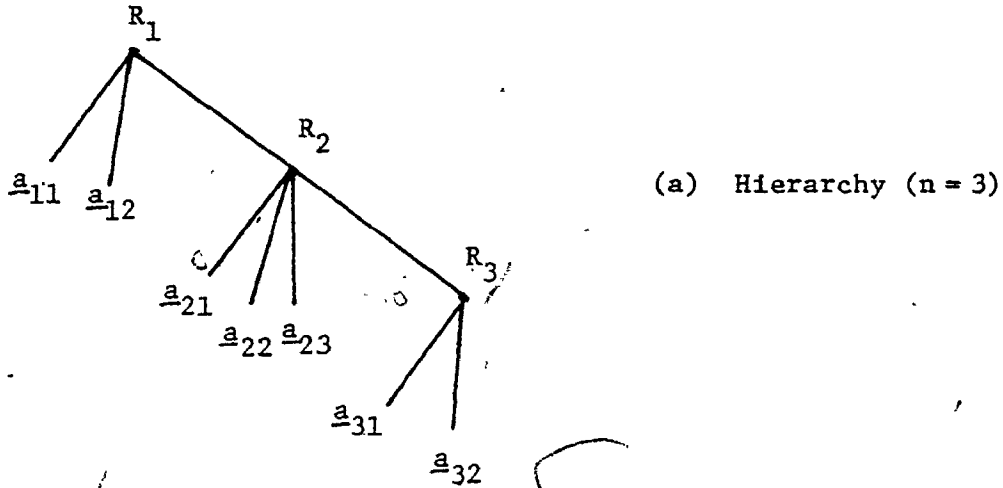
In WCRC, the hierarchical structure would be stored as a number of PCP's. An example, a hierarchy with three levels of embedding, is shown in Fig. 6.3. Corresponding to relation  $R_i$  there will be  $m_i$  PCP's of type  $[\underline{\alpha}_i; \underline{a}_{ij}]$  and  $(i-1)$  PCP's of type  $[\underline{\alpha}_i; \underline{\alpha}_j]$ . Note  $\underline{\alpha}_i$  is the key in  $R_i$ . Therefore, PCP's  $[\underline{\alpha}_i; \underline{a}_{ij}]$  and  $[\underline{\alpha}_i; \underline{\alpha}_j]$  would be a set of EWCR's each. Also in  $R'_n$ , a system defined key  $\underline{\alpha}_n$  is introduced because the key would be repeated several times in the PCP's. Since keys are encoded, introducing  $\underline{\alpha}_n$  would minimize the storage required. Now, the storage requirement for WCRC with PCP - option I can be obtained as follows.

Let  $s_i$  be the average size of an EWCR in PCP's corresponding to relation  $R_i$ . The storage required ( $S_v$ ) for the values  $\underline{a}_{ij}$  of all domains in all PCP's of type  $[\underline{\alpha}_i; \underline{a}_{ij}]$  corresponding to relations  $R_1$  to  $R_n$  is given by

$$S_v = \sum_{i=1}^n \sum_{j=1}^{m_i} a_{ij} \frac{\ell_i}{s_i} \quad (6.4)$$

The storage need ( $S_K$ ) for key values ( $\underline{\alpha}_i$ 's) in all PCP's of type  $[\underline{\alpha}_i; \underline{a}_{ij}]$  is given by

$$S_K = \sum_{i=1}^n \alpha_i \ell_i m_i \quad (6.5)$$



(b) The ANF Form

$$R_1 \quad (\underline{a}_{11}, \underline{a}_{12}, \underline{\alpha}_1)$$

$$R_2 \quad (\underline{a}_{21}, \underline{a}_{22}, \underline{a}_{23}, \underline{\alpha}_1, \underline{\alpha}_2)$$

$$R_3 \quad (\underline{a}_{31}, \underline{a}_{32}, \underline{\alpha}_1, \underline{\alpha}_2, \underline{\alpha}_3)$$

(c) The PCP's

$(\underline{\alpha}_i - \underline{a}_{ij})$	$[\underline{\alpha}_1; \underline{a}_{11}]$	$[\underline{\alpha}_2; \underline{a}_{21}]$	$[\underline{\alpha}_3; \underline{a}_{31}]$
type	$[\underline{\alpha}_1; \underline{a}_{12}]$	$[\underline{\alpha}_2; \underline{a}_{22}]$	$[\underline{\alpha}_3; \underline{a}_{32}]$
		$[\underline{\alpha}_2; \underline{a}_{23}]$	

$(\underline{\alpha}_i - \underline{\alpha}_j)$		$[\underline{\alpha}_2; \underline{\alpha}_1]$	$[\underline{\alpha}_3; \underline{\alpha}_2]$	$[\underline{\alpha}_3; \underline{\alpha}_1]$
type				

Figure 6.3 An Example Showing How ANF is Stored as PCP's

The storage required ( $s_p$ ) for address (pointer) fields in one PCP of type  $[\underline{\alpha}_i; \underline{a}_{ij}]$  is given by

$$s_p = 2 \cdot \ell_i \cdot v_i \quad (6.6)$$

where  $v_i$  is the storage for one pointer field

and  $2\ell_i$  is the number of forward and backward pointers in both halves of the track(s)

The total storage ( $S_p$ ) need for pointers in all PCP's of type  $[\underline{\alpha}_i; \underline{a}_{ij}]$  is given by

$$S_p = \sum_{i=1}^n 2\ell_i \cdot v_i \cdot m_i \quad (6.7)$$

The storage need for PCP's of type  $[\underline{\alpha}_i; \underline{\alpha}_j]$  can be obtained as follows. Assume that the average size of EWCR's is still  $s_i$  in these PCP's. The total storage need ( $s_{\alpha_k}$ ) for pointers in all the PCP's of type  $[\underline{\alpha}_i; \underline{\alpha}_j]$  is given by

$$S_{\alpha p} = \sum_{i=1}^n (i-1) 2\ell_i \cdot v_i \quad (6.8)$$

The storage required by the nonkey and key values of  $[\underline{\alpha}_i; \underline{\alpha}_j]$  is obtained as follows. Recall, at level  $i$ ,  $\alpha_i$  is the key.

For level 1, the storage for values = 0

For level 2, the storage for values =  $\alpha_2^{\ell_2} + \frac{\alpha_1^{\ell_1}}{s_1}$

For level 3, the storage for values =  $\alpha_3^{\ell_3} + \frac{\alpha_2^{\ell_2}}{s_2} + \frac{\alpha_1^{\ell_1}}{s_1}$

so on

$$= 2\alpha_3^{\ell_3} + \frac{\alpha_1^{\ell_1}}{s_1} + \alpha_2 \frac{\ell_2}{s_2}$$

For level n, storage for values =  $(n-1) \alpha_n^{\ell_n} + \alpha_1 \frac{\ell_1}{s_1} \dots$

$$\alpha_{n-1} \frac{\ell_{n-1}}{s_{n-1}}$$

Summing up these terms, the total storage for values is  $S_{\alpha\alpha}$  is given by

$$\begin{aligned} S_{\alpha\alpha} &= \sum_{n=2}^n (i-1) \alpha_i^{\ell_i} + (n-1) \frac{\alpha_1^{\ell_1}}{s_1} + (n-2) \frac{\alpha_2^{\ell_2}}{s_2} + \dots + 1 \alpha_{n-1} \frac{\ell_{n-1}}{s_{n-1}} \\ &= \sum_{i=2}^n (i-1) \alpha_i^{\ell_i} + \sum_{i=1}^{n-1} (n-i) \alpha_i \frac{\ell_i}{s_i} \end{aligned}$$

The total storage for PCP - option I,  $S_3$  is given by

$$\begin{aligned} S_3 &= S_v + S_k + S_p + S_{ap} + S_{\alpha\alpha} \\ &= \sum_{i=1}^n \sum_{j=1}^{m_i} a_{ij} \frac{\ell_i}{s_i} + \sum_{i=1}^n \alpha_i^{\ell_i} m_i + \sum_{n=1}^n 2\ell_i v_i (m_i + i - 1) \\ &\quad + \sum_{i=2}^n (i-1) \alpha_i^{\ell_i} + \sum_{i=1}^{n-1} (n-i) \frac{\alpha_i \ell_i}{s_i} \end{aligned} \tag{6.9}$$

Equation 6.9 can be further simplified making the following assumptions:

- 1) On the average, all  $s_i$ 's are about the same and  $s_i = s \cdot v_i$
- 2) All relations have the same length on the average given by  $l_i = l \cdot v_i$ ,  $\therefore v_i = v \cdot i$
- 3) The storage required for all domain values,  $a_{ij}$  is the same, i.e.,  $a_{ij} = a \cdot v_{i,j}$

$$S_3 = \frac{1}{s} a l \left( \sum_{i=1}^n m_i \right) + \alpha l \left( \sum_{i=1}^n m_i \right) + 2lv \cdot \left( \sum_{i=1}^n (m_i + i - 1) \right) \\ + \alpha l \sum_{i=2}^n (i-1) + \frac{\alpha l}{s} \sum_{i=1}^{n-1} (n-i) \quad (6.10)$$

Let  $\sum_{i=1}^n m_i = N_D$ , the total number of domains of all the relations

Then, Eq. 6.10 becomes,

$$S_3 = \frac{1}{s} a l N_D + \alpha l N_D + 2lv \left( N_D + \sum_{i=1}^n (i-1) \right) + \alpha l \sum_{i=2}^n (i-1) \\ + \frac{\alpha l}{s} \sum_{i=1}^{n-1} (n-i) \\ = \frac{1}{s} a l N_D + \alpha l N_D + 2lv \left( N_D + \frac{n(n-1)}{2} \right) + \alpha l \frac{n(n-1)}{2} + \frac{\alpha l}{s} \frac{n(n-1)}{2}$$

Now,  $v = \lceil \log_2 l \rceil = \alpha$  ( $\alpha$  is the number of distinct values in a PCP)

Eq. 6.11 becomes:

$$\begin{aligned}
 S_3 &= \frac{1}{s} a\ell N_D + \alpha\ell N_D + 2\alpha\ell \left(N_D + \frac{n(n-1)}{2}\right) + \alpha\ell \frac{n(n-1)}{2} \\
 &\quad + \frac{\alpha\ell}{s} \frac{n(n-1)}{2} \\
 &= \frac{1}{s} a\ell N_D + \alpha\ell \left[3N_D + \frac{3n(n-1)}{2} + \frac{n(n-1)}{2s}\right] \quad (6.12)
 \end{aligned}$$

Applying the same assumptions 1-3, to Equations 6.2 and 6.3 we get the following:

$$S_1 = a\ell N_D + \alpha\ell (n-1) + \alpha\ell \frac{n(n-1)}{2} + \ell \sum_{i=1}^n R_i$$

Note  $R_i$ , the storage required in bits for the relation number  $i$ , is  $\lceil \log_2 n \rceil$ . Let  $\beta$  denote  $\lceil \log_2 n \rceil$ .

$$S_1 = a\ell N_D + \alpha\ell \left[(n-1) + \frac{n(n-1)}{2}\right] + n\beta\ell \quad (6.13)$$

$$S_2 = a\ell N_D + \alpha\ell \left[(n-1) + \frac{n(n-1)}{2}\right] \quad (6.14)$$

The storage requirement for PCP - option II can be obtained similar to option I as follows:

The storage needed for  $[\underline{\alpha}_i; \underline{\alpha}_{ij}]$  type PCP's,  $s_{\alpha a}$  is given by

$s_{\alpha a}$  = storage for key values of all PCP's + storage for monkey values of all PCP's

$$= \sum_{i=1}^n \alpha_i \ell_i m_i + \sum_{i=1}^n \sum_{j=1}^{m_i} a_{ij} \frac{\ell_i}{s_i} \quad (6.15)$$

Storage required for  $[\underline{\alpha}_i ; \underline{\alpha}_j]$  type PCP's,  $s_{\alpha\alpha}$  is given by:

$$s_{\alpha\alpha} = \sum_{i=2}^n (i-1) \alpha_i \ell_i + \sum_{i=1}^{n-1} (n-i) \frac{\alpha_i \ell_i}{s_i} \quad (6.16)$$

Note, there are no pointer fields in the option II.

The total storage for PCP - option II,  $S_4$  is given by:

$$\begin{aligned} S_4 &= s_{\alpha a} + s_{\alpha\alpha} \\ &= \sum_{i=1}^n \alpha_i \ell_i m_i + \sum_{i=1}^n \sum_{j=1}^{m_i} a_{ij} \frac{\ell_i}{s_i} + \\ &\quad \sum_{i=2}^n (i-1) \alpha_i \ell_i + \sum_{i=1}^{n-1} (n-i) \frac{\alpha_i \ell_i}{s_i} \end{aligned} \quad (6.17)$$

Based on assumption 1-3, Eq. 6.17 can be further reduced to the following form:

$$S_4 = \frac{1}{s} a \ell N_D + \alpha \ell \left[ N_D + \frac{n(n-1)}{2} \left( 1 + \frac{1}{s} \right) \right] \quad (6.18)$$

While deriving Equations 6.12 and 6.18 it is assumed that the average EWCR size ( $s$ ) is equal to the physical EWCR size on the track. ( $s_t$ ). However, if they are not equal, the track EWCR size is so chosen that  $s$  is an integral multiple of  $s_t$ . Taking this factor into account, the Equations 6.12 and 6.18 will be modified, as follows:



$$S_3 = \frac{s_e}{s} \alpha l N_D + \alpha l \left[ 3N_D + \frac{3(n-1)n}{2} + \frac{n(n-1)}{2s} s_e \right] \quad (6.19)$$

$$S_4 = \frac{s_e}{s} \alpha l N_D + \alpha l \left[ N_D + \frac{n(n-1)}{2} \left( 1 + \frac{s_e}{s} \right) \right]$$

$$\text{where } s_e = \left\lceil \frac{s}{s_t} \right\rceil \quad (6.20)$$

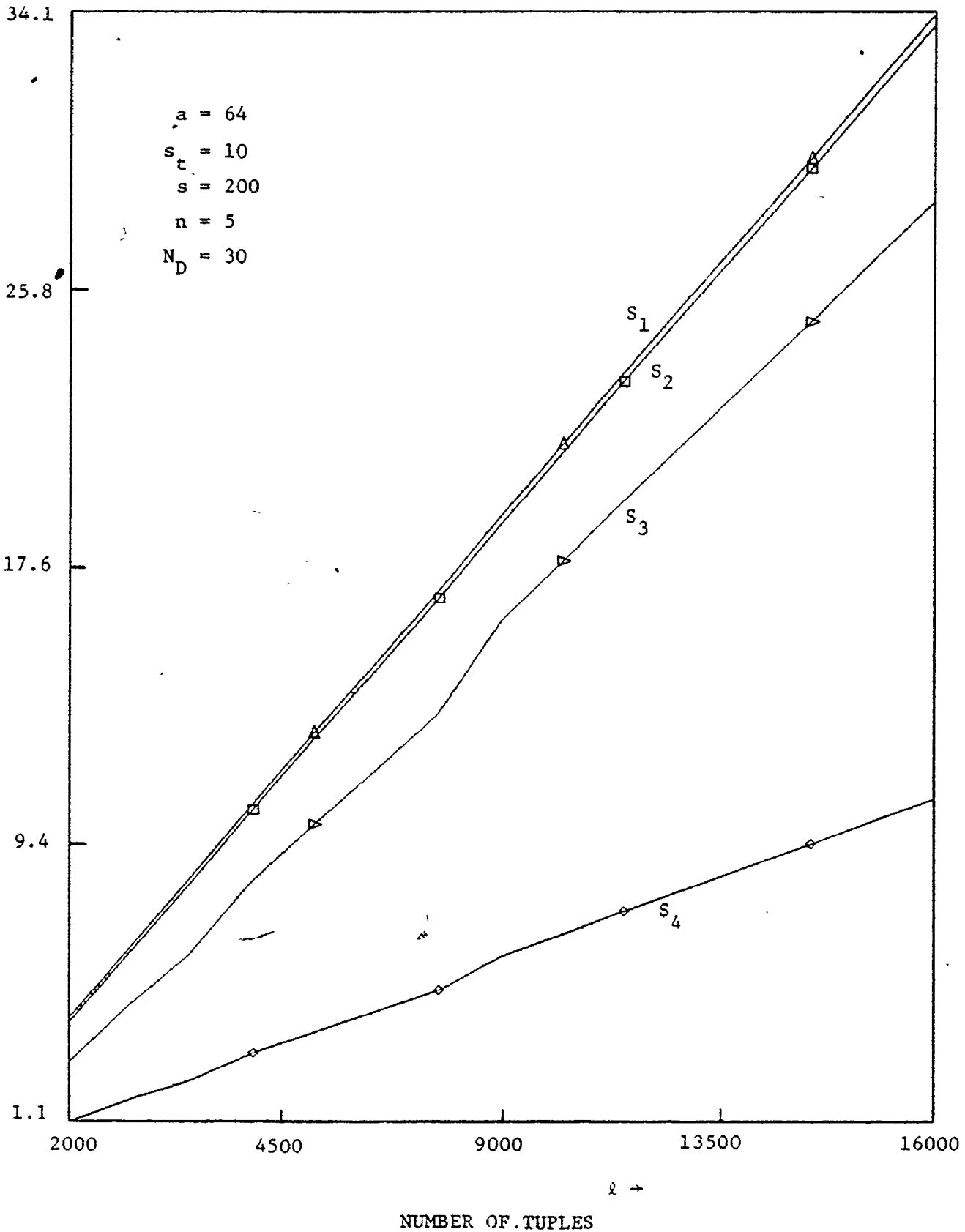
For Equations 6.13, 6.14, 6.19 and 6.20, the variations of  $S_1$ ,  $S_2$ ,  $S_3$  and  $S_4$  w.r.t.  $l$ ,  $N_D$ ,  $n$ ,  $s$ ,  $a$  are shown graphs 6.1, 6.2, 6.3, 6.4 and 6.5.

Graph 6.1: This shows that the storage requirement is less for WCRC when compared to DeFiore's scheme or GDBMS scheme. Among options I and II, option II has the least storage requirement.

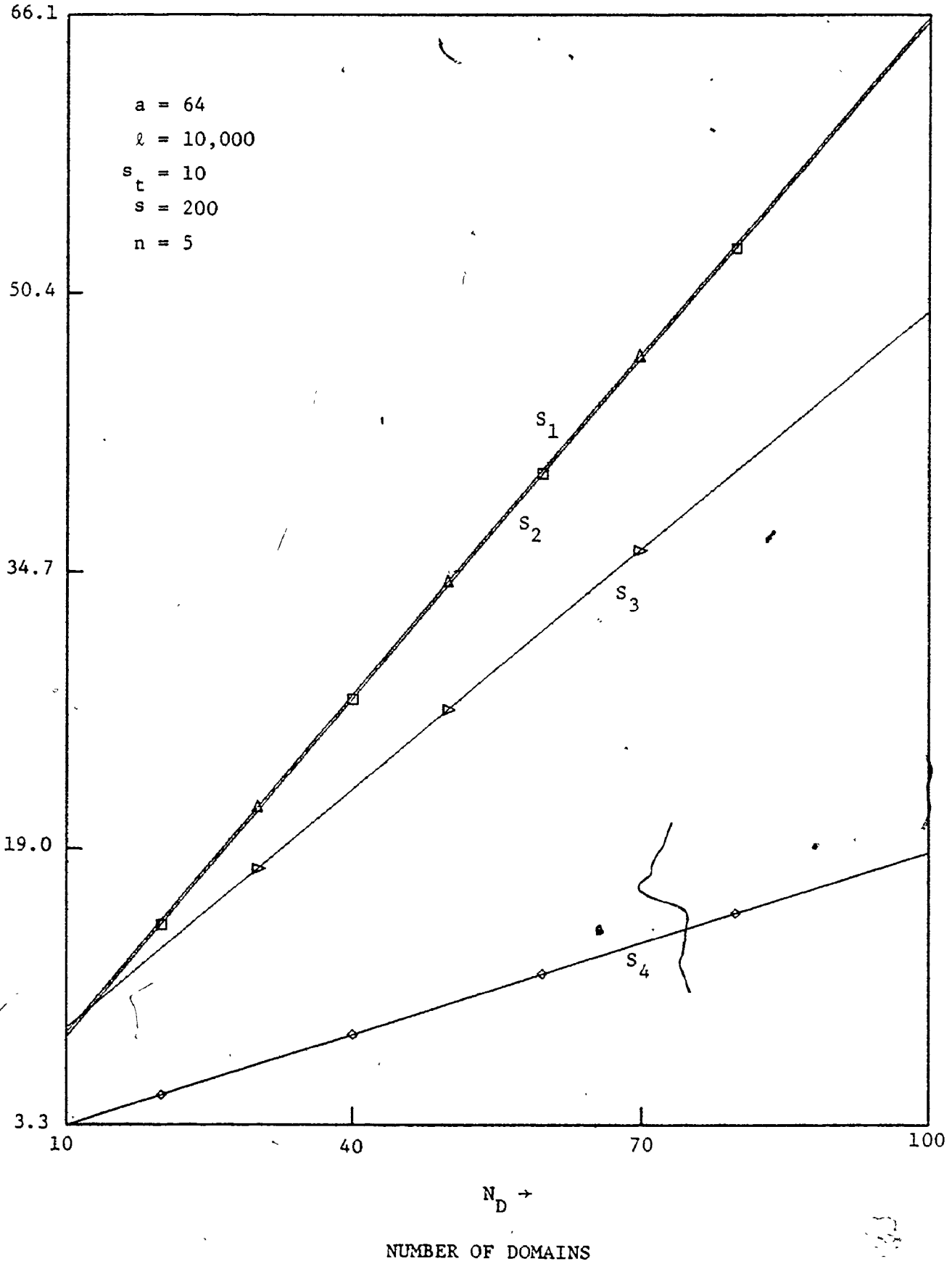
Graph 6.2: This shows the variations of  $S_1$ ,  $S_2$ ,  $S_3$  and  $S_4$  w.r.t.  $N_D$  while  $l$ ,  $n$ ,  $s$  are kept constant. Once again, WCRC storage scheme takes less storage and WCRC - option II takes the least amount of storage.

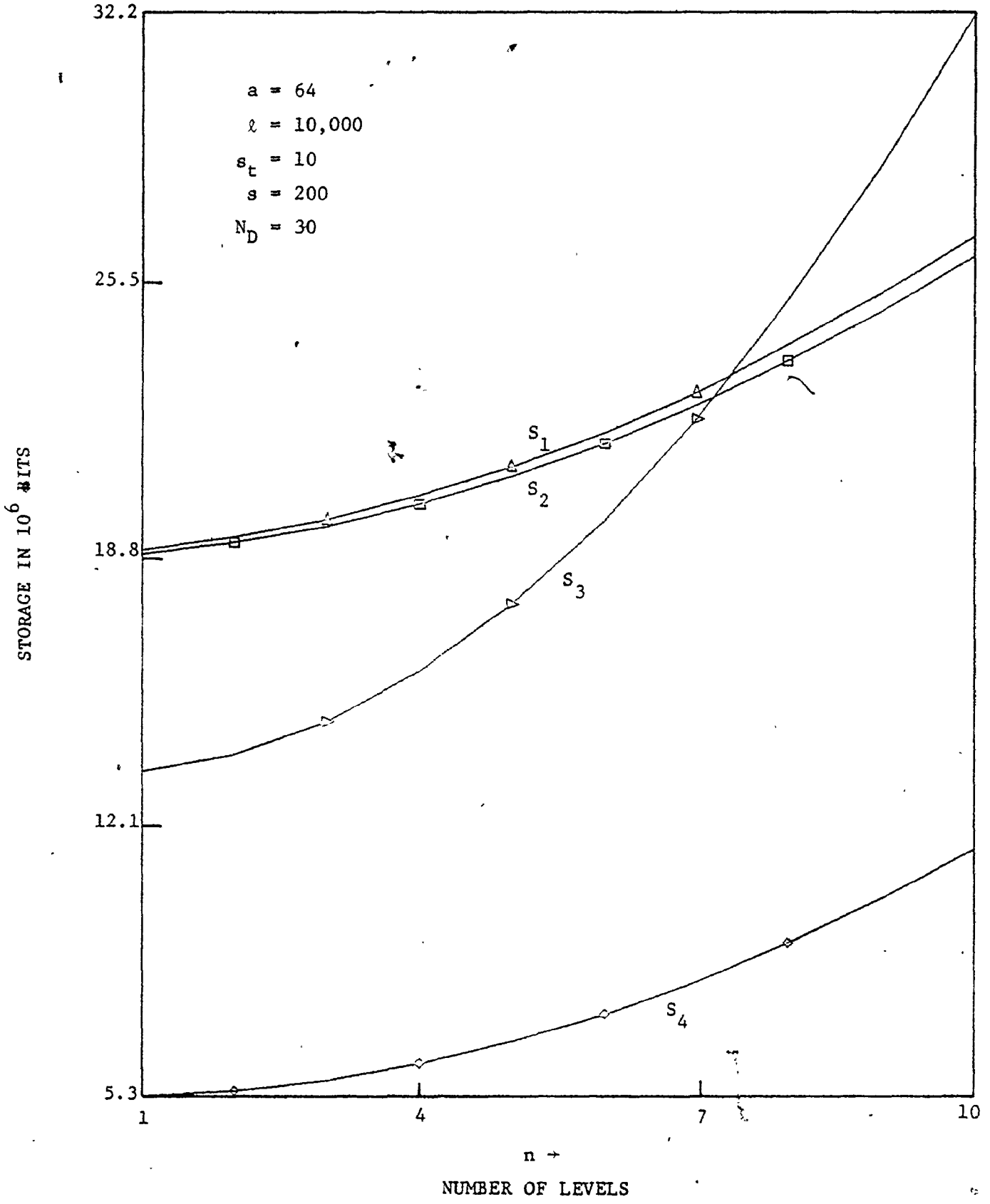
Graph 6.3: This shows the variations of  $S_1$ ,  $S_2$ ,  $S_3$  and  $S_4$  w.r.t.  $n$ . WCRC - option II needs the least storage among all. The storage for option I gets worse at higher values of  $n$ , the number of levels.

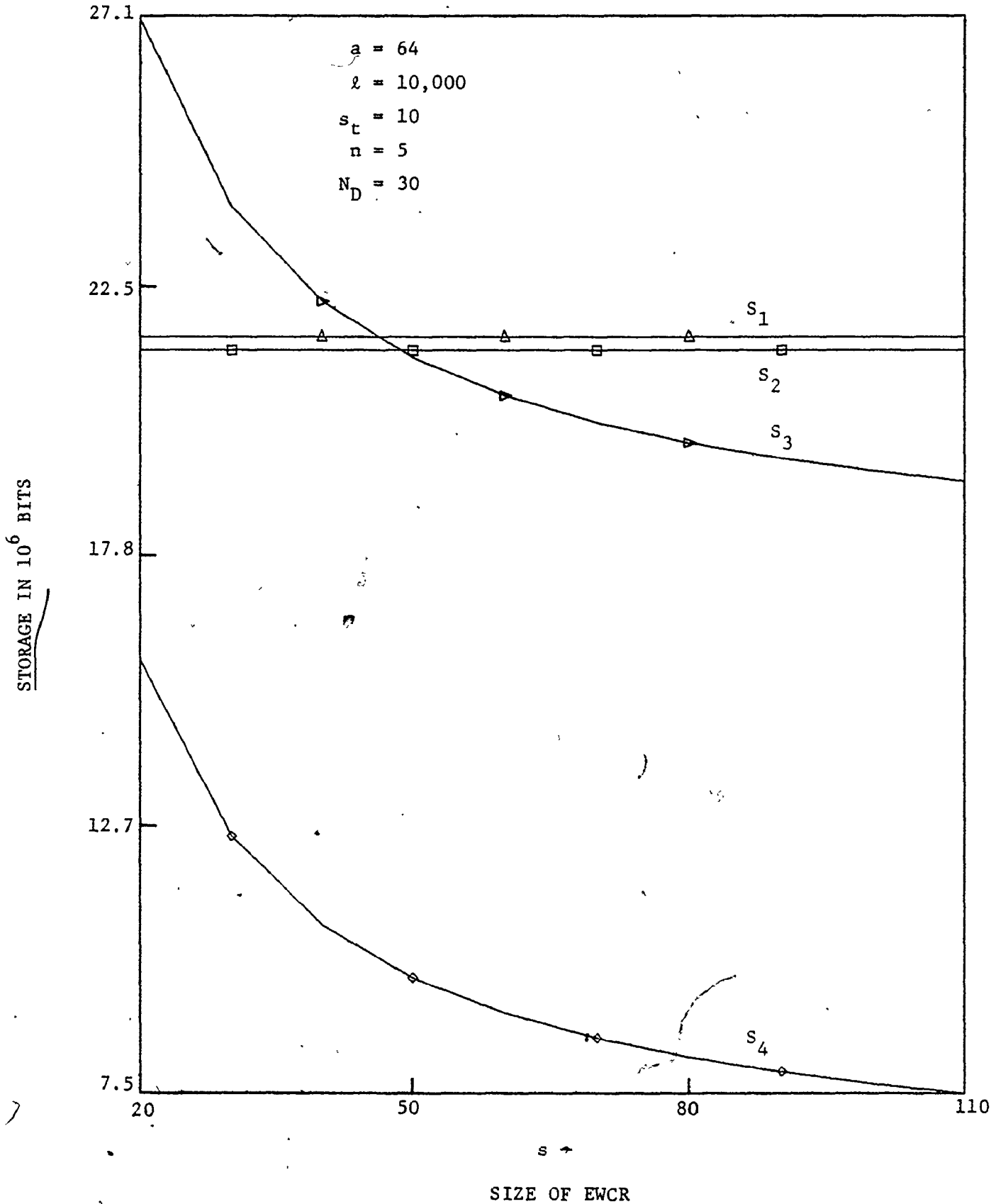
Graph 6.4: This graph shows how the storage of different schemes vary with the EWCR size. Note  $S_1$  and  $S_2$  remain constant throughout,



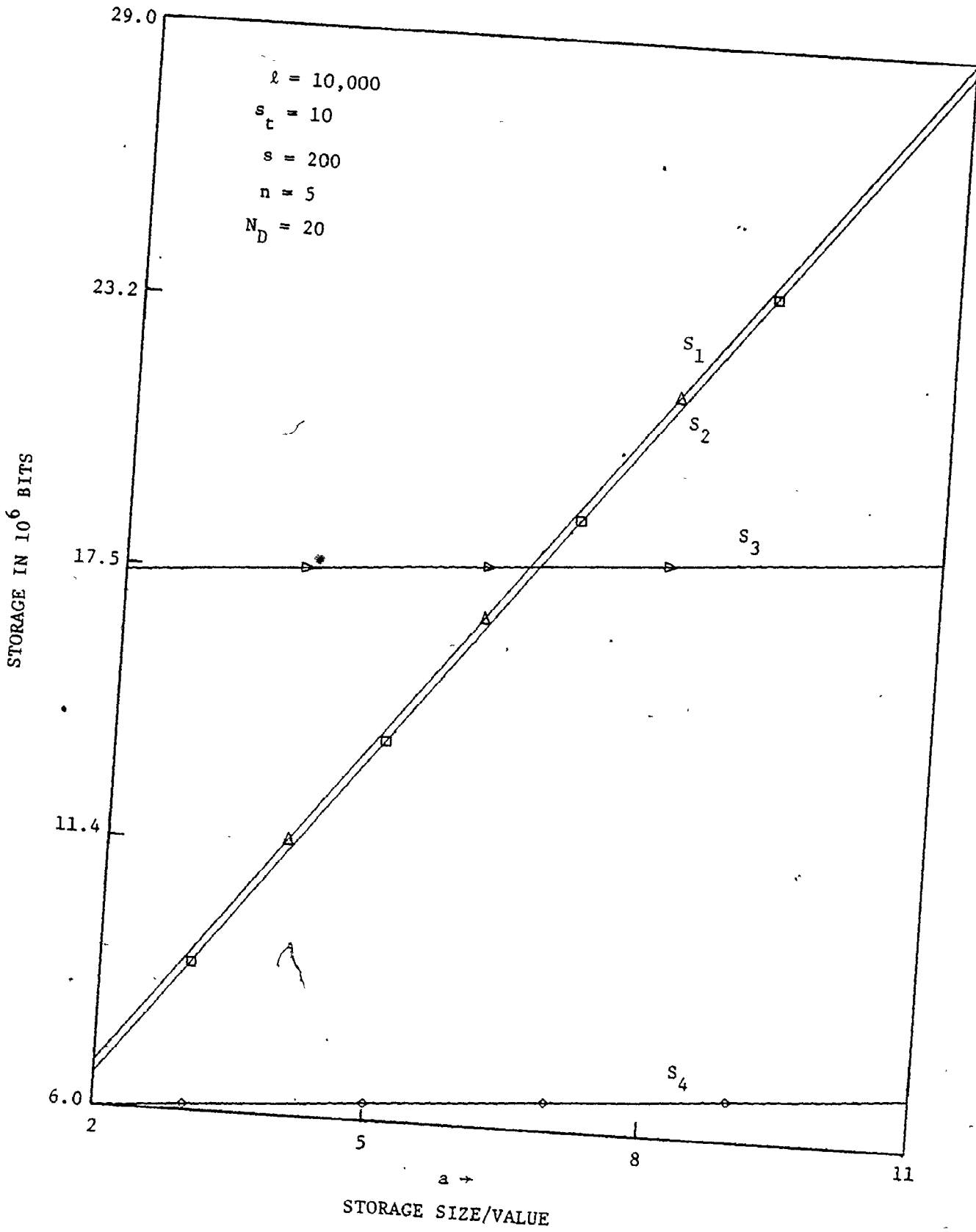
Graph 6.2 STORAGE vs.  $N_D$







Graph 6.5 STORAGE vs. a



while  $S_3$  and  $S_4$  decrease with increasing  $s$  value. It is clear that, for higher values of  $s$  the storage for  $S_3$  and  $S_4$  decrease further. WCRC - option II always takes the least amount of storage.

Graph 6.5: This shows variations of  $S_1$ ,  $S_2$ ,  $S_3$  and  $S_4$  with the storage size of nonkey domain value,  $a$ . As can be seen easily, it is economical to use PCP storage option I for storage sizes greater than about '36 bits' (for the given values of  $\ell$ ,  $n$ ,  $N_D$ , and  $s$ ). But option II provides the least storage structure for all sizes of the nonkey domain values.

### 6.3 Retrieval Time

The following general case of retrieval is chosen to compare the performance of three schemes. The retrieval query involves qualification over all the attributes of the hierarchy, starts at the top of the hierarchy and continues through all levels of embedding. In this section, only marking time for qualifications is considered. Also, since RAP [OZKARAHAN 76] does not allow interattribute qualifications, only simple qualifications of the type  $\langle \text{attribute} \rangle \langle \text{opr} \rangle \langle \text{constant} \rangle$  are considered. The overall retrieval times for GDBMS and WCRC are examined in the next section.

Let  $c_{ij}$  represent the number of search criteria on  $a_{ij}$  and  $|p_i|$  denote the number of tuples which respond in  $R_i$  to the search criterion. The total number of interrogations ( $T_1$ ) needed by the query in DeFiore's scheme [DEFIORE 74] is given by:

$$T_1 = \sum_{i=1}^n \sum_{j=1}^{m_i} c_{ij} + \sum_{i=1}^{n-1} |p_i| \quad (6.21)$$

This is described as the worst case time, since logical operations between domains of the same relation can be performed in parallel in the associative memory.

The number of interrogations in the case of GDBMS, which uses RAP [OZKARAHAN 75] at the internal level, can be derived as follows. If  $c_{ij}$  is the number of search criteria on  $a_{ij}$  attributes of  $R_1$ , the number of interrogations ( $t$ ) needed on  $R_1$  is given by,

$$t = \frac{1}{k} \sum_{j=1}^{m_i} c_{ij}, \quad k \text{ is the RAP parameter}$$

Note, in RAP only  $k$  domains can participate at one time in the qualification. Let  $|p_1|$  be the number of tuples which respond to the above search on  $R_1$ . The  $\alpha_1$  values from these tuples are used as arguments to search for the matching  $\alpha_1$  values in the relation  $R_2$ . This would take  $1 + \frac{|p_1|}{k} + Y_1$  where  $Y_1$  is the number of cells with marked tuples [OZKARAHAN 76]. This can be generalized for  $n$  levels on similar grounds.

The total number of interrogations,  $T_2$  for  $n$ -levels of embedding for GDBMS is given by:

$$T_2 = \frac{1}{k} \sum_{i=1}^n \sum_{j=1}^{m_i} c_{ij} + \sum_{i=1}^{n-1} \left(1 + \frac{|p_i|}{k} + Y_i\right) \quad (6.22)$$

For the sake of simplicity, we assume that on the average, every cell



corresponding to  $R_i$  has marked tuples.

The number of interrogations for the same search criteria in WCRC can be derived as follows.

Since all the cells are independent in PCP schemes, the search on all cells can be performed in parallel. However, each cell has only  $m$  comparators, which would restrict the number of search criteria on each cell. The maximum number of interrogations for search on the domains is given by  $\max \left[ \frac{c_{ij}}{K} \right]$ , where  $K$  is the WCRC parameter. In these interrogations all the attribute domains in all PCP's of levels 1 to  $n$  are marked in parallel. Their corresponding key values ( $\alpha_i$  values) are also marked in the same revolution. (Note, in case of RAP and WCRC, the words 'number of interrogations' and 'number of revolutions' are used interchangeably). Depending upon the qualification of the query, a boolean evaluation (AND/OR combination) must be performed on the marked  $\alpha_i$  key values at level  $i$  and these should, in turn, be used to mark the PCP  $[\alpha_{i+1}, \alpha_i]$  at the next level  $i+1$ . This may be called traverse time along the hierarchy. The traverse time can be obtained as follows.

Traverse Time ( $t_t$ ):

Let  $L_i$  be the average number of tracks occupied by each PCP at level  $i$ . So at level  $i$ , we have  $m_i$  PCP's each occupying  $L_i$  tracks which are of the type  $[\alpha_i, \alpha_{ij}]$  where  $j = 1, 2, \dots, m_i$ .

During search and mask time, all the attributes  $\alpha_{ij}$  ( $j=1, \dots, m$ ) satisfying  $c_{ij}$  search criteria and the corresponding  $\alpha_i$  values are

marked. Now, the boolean expression involving  $c_{ij}$  is evaluated over these  $\alpha_i$  values, using the special hardware as follows (Fig. 6.4).

(1) Note, each cell/track has an associated RAM with the number of bits same as that of the number of distinct key values on it. All these cell RAM's corresponding to a PCP would form a "PCP RAM". A PCP RAM is accessible by any of its tracks. Now, using the marked  $\alpha_i$  values, the corresponding RAM bits are marked in all PCP's at level  $i$ . These PCP RAM's at level  $i$  are used as the inputs to the Boolean Evaluator which performs the required boolean expression and writes back the result on one of the PCP RAM's, say  $RM_{i1}$ .

The time taken to mark the PCP RAM's,  $t_{mr}$  is given by,

$$t_{mr}(i) = 1 \text{ revolution} \quad (1)$$

Note, all PCP RAM's are marked in parallel. However, in option I, collisions are possible in theory because of possible repeated key values at the track boundaries of a PCP. The chances of a collision are remote because the repeated key values are at the end of  $i^{\text{th}}$  track and the beginning of  $(i+1)^{\text{th}}$  track and these opposite ends are not processed at the same time. This problem is nonexistent in option II.

The time taken to evaluate boolean expression  $t_b$  is given by:

$$t_b(i) = m \cdot k_i \cdot t_c \text{ units}$$

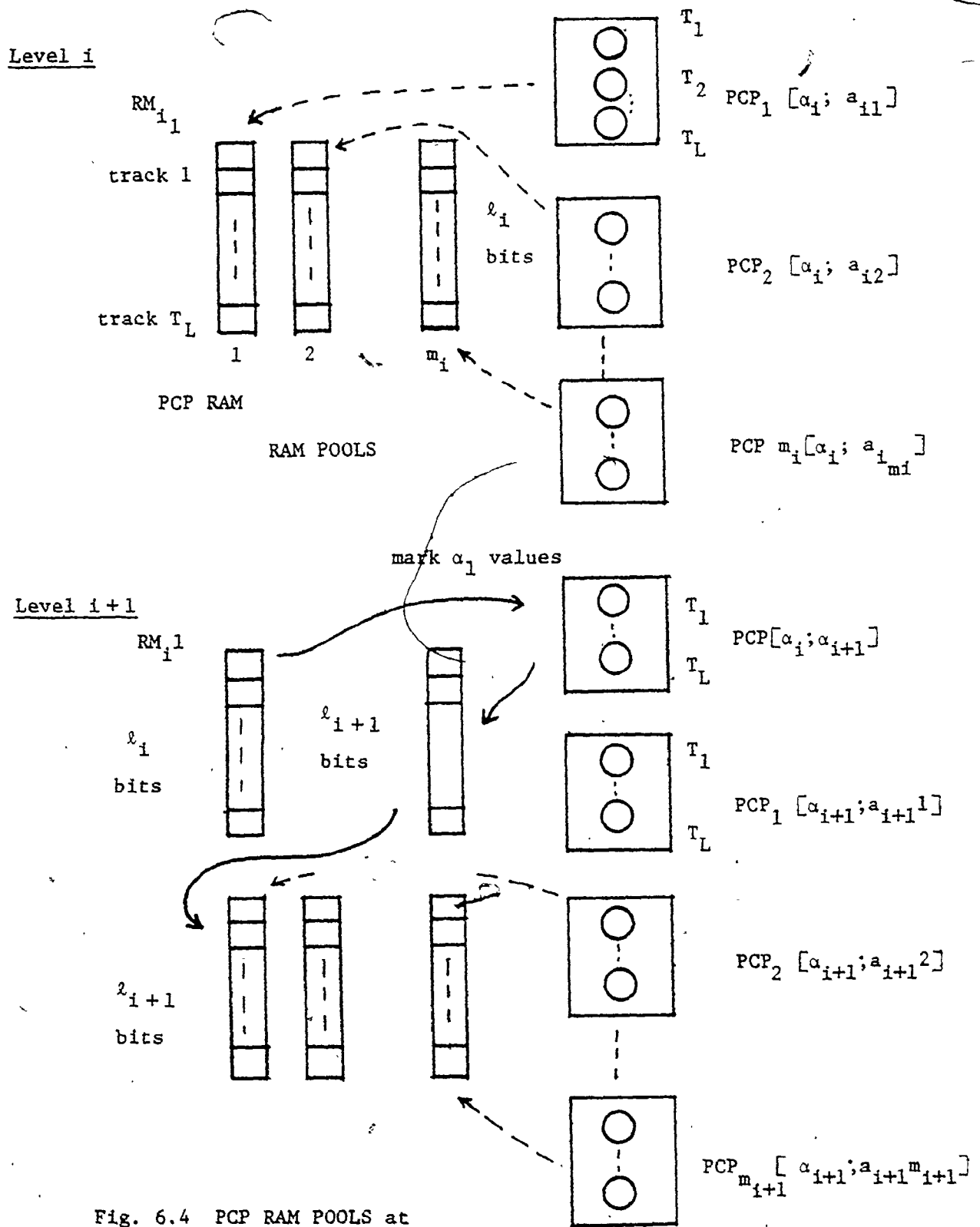


Fig. 6.4 PCP RAM POOLS at Level i and I+1.

where  $m$  = number of terms in the boolean expression

$l_i$  = number of bits in each PCP RAM at level  $i$ ,

$t_c$  = clock rate of the Boolean Evaluator

Since  $t_c$  could be in the order of manoseconds,  $t_b$  may be neglected compared to the revolution time (0.1 to 0.5 seconds).

(2) Now, the RAM  $RM_{i1}$  at level  $i$  is used to mark the  $\alpha_i$  values in the PCP  $[\alpha_i, \alpha_i + 1]$  at level  $i+1$ . Marking is done by accessing the various units of  $RM_{i1}$  through the RAM bus under the control of Q.P. For each value of  $\alpha_i$  and marking it if its corresponding RAM bit is set. The corresponding  $\alpha_{i+1}$  values that go with the  $\alpha_i$  values are also set in the process.

The time taken to link level  $i$  and  $i+1$ ,  $t_e$  is given by

$$t_e(i) = L_i \text{ revolutions} \quad (2)$$

Because there is only one RAM bus which can be used by one cell at a time.

(3) The marked  $\alpha_{i+1}$  values are used in turn to mark the corresponding PCP RAM as explained earlier. This again takes another revolution ( $t_{rm}$ ).

$$\text{i.e. } t_{rm}(i) = 1 \text{ revolution} \quad (3)$$

The contents of the marked PCP RAM pool at level  $i+1$  are transferred

onto all the PCP RAM pools at that level. This transfer time is negligible compared to the revolution time.

(4) Finally, the PCP RAM's are used to mark the  $\alpha_{i+1}$  values in all the PCP's, which takes one revolution ( $t_{tr}$ ).

These steps (1-4) are repeated at level  $\alpha_{i+1}$ . Thus, the time taken to traverse from level  $i$  to level  $i+1$ ,  $t_t(i)$  is given by,

$$\begin{aligned} t_t(i) &= t_{mr}(i) + t_e(i) + t_{rm}(i) + t_{tr}(i) \\ &= 3 + L_i \end{aligned}$$

The traverse time for the whole hierarchy,

$$t_t = \sum_{i=1}^{n-1} L_i + 3(n-1)$$

Assuming the number of tracks occupied by all PCP's at all levels to be  $L$  on average, we get

$$t_t = (n-1)(3+L)$$

The total number of interrogations,  $T_3$  for marking the qualification of the query over the entire hierarchy is given by:

$$T_3 = \max \left( \frac{c_{ij}}{K} \right) + (n-1)(3+L) \quad (6.23)$$

If option II, the number of interrogations would be the same as option I, because marking (SET instruction) takes the same amount

of time in both. To compare the retrieval times of three schemes, the following assumptions are made.

- 1) The number of search criteria on each domain is chosen to be 1.
- 2) The average number of selected tuples at all levels is assumed to be the same given by  $|p_i| = p$  for all  $i$ .
- 3) The number of tracks occupied each relation in RAP is taken to be the same for all relations, i.e.  $Y_i = y$  for all  $i$ . The same is true for the PCP.

Based on these assumptions, the equations 6.21, 6.22 and 6.23 would become:

$$T_1 = N_D + (n-1) p \quad (6.24)$$

$$T_2 = \frac{N_D}{k} + (n-1) \left(1 + \frac{p}{k} + y\right) \quad (6.25)$$

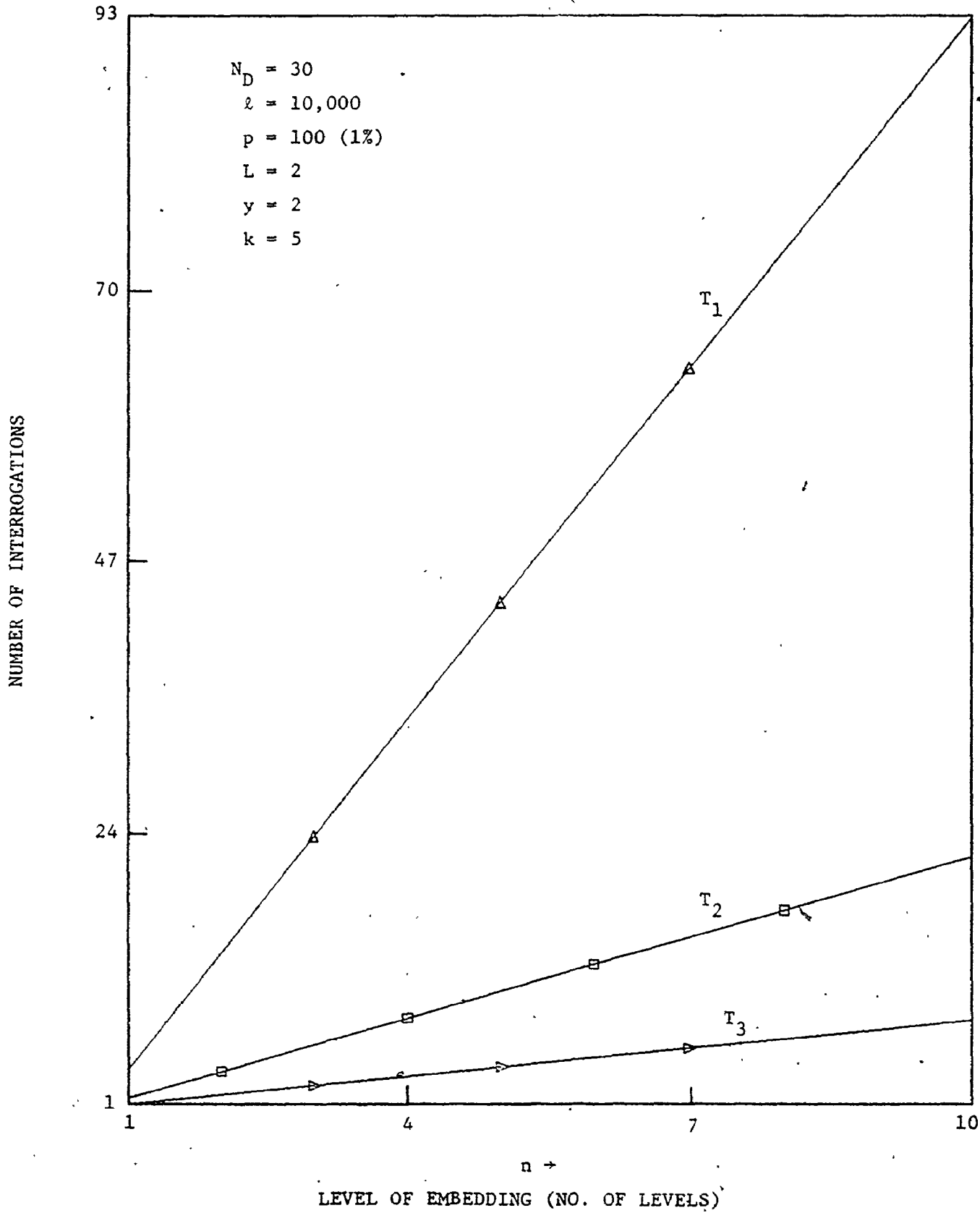
$$T_3 = 1 + (3+L) (n-1) \quad (6.26)$$

be variations of  $T_1$ ,  $T_2$  and  $T_3$  (Equations 6.24, 6.25 and 6.26) w.r.t.  $n$ , the level of embedding is shown in graph 6.6. It is clear from this graph that WCRC has the least number of interrogations of the three schemes.

#### 6.4 Comparison of Overall Retrieval Times of GDBMS and WCRC

In this section, both marking time and I/O time for a general retrieval query are examined and a comparison of GDBMS and WCRC based on this overall time is presented. For the sake of simplicity, the

Graph 6.6 NUMBER OF INTERROGATIONS  
vs. LEVEL OF EMBEDDING



following assumptions are made:

- 1) There is a qualification (selection) clause on every attribute of each relation.
- 2) At each level, i.e. in each relation, one attribute is chosen for output.
- 3) All levels are joined (natural) over  $q_1$  values.
- 4) The qualification is of the type  $\langle \text{attribute} \rangle \langle \text{opr} \rangle \langle \text{constant} \rangle$ , where  $\langle \text{opr} \rangle$  is one of ( $=$ ,  $\neq$ ,  $>$ ,  $<$ ,  $\geq$ ,  $\leq$ )

Note, GDBMS, does not allow interattribute qualifications.

#### GDBMS

As shown in the previous section, the total marking time, ( $t_m$ ) would be

$$\frac{N_D}{k} + (n-1) (1 + p/k + y)$$

where  $N_D$  is the total number of domains/attributes.

Once all the relations are marked, each relation is projected over one domain, which has to be output. The time taken for one such projection in RAP is shown in Appendix C. It is given by:

$$t_{p1} = 1 + 2 n_d$$

where  $n_d$  is the number of distinct values in the domain of interest.

Total time for  $n$  projection ( $t_p$ ) =  $n \cdot (1 + 2 n_d)$



Now, all the relations have to be joined on the marked  $\alpha_1$  values. The time taken for one such physical join is given in Appendix C. To physically join  $R_n$  with, say,  $R_i$  would take  $t_{j1}$  revolutions where  $t_{j1}$  is given by:

$$t_{j1} = p_n (2 + 4 q_i) + 3 p_n + 2 \text{ revolutions}$$

where  $p_n$  = number of qualified tuples in  $R_n$

$q_i$  = number of tuples that qualify for join condition in  $R_i$  for every tuple in  $p_n$

Similarly, the number of revolutions ( $t_{j2}$ ) to join  $R_n$  with  $R_i$  and  $R_j$  is given by:

$$t_{j2} = p_n (2 + 4(q_i + q_j)) + 3 p_n \cdot 2 + 2$$

The total number of revolutions ( $t_j$ ) to join  $R_n$  with  $R_1$  to  $R_{n-1}$  is given by:

$$t_j = p_n (2 + 4 \sum_{i=1}^{n-1} q_i) + 3 p_n (n-1) + 2$$

The overall time ( $t_t$ ) for retrieval =  $t_m + t_p + t_j$

$$\begin{aligned} &= \frac{N_D}{k} + (n-1) (1 + \frac{p}{k} + y) + n (1 + 2 n_d) \\ &+ p_n (2 + 4 \sum_{i=1}^{n-1} q_i) + 3 p_n (n-1) + 2 \end{aligned}$$

Assuming  $p_n = p$ , we get

$$t_{t1} = \frac{N_D}{k} + (n-1) \left(1 + \frac{p}{k} + y\right) + n(1 + 2n_d) + p \left(2 + 4 \sum_{i=1}^{n-1} q_i\right) + 3p(n-1) + 2 \quad (6.27)$$

For WCRC, the overall time can be derived as follows. The marking and traverse time, as shown in the previous section, would be  $1+(3+L)(n-1)$  revolutions. After marking, we have the following situation. There are  $n$  PCP's  $[\underline{\alpha}_1; \underline{\alpha}_2]$ ,  $[\underline{\alpha}_1; \underline{\alpha}_3]$ ,  $\dots$ ,  $[\underline{\alpha}_1; \underline{\alpha}_n]$ , all of them with their  $(\underline{\alpha}_1; \underline{\alpha}_i)$  value pairs marked. They have to be joined based on the marked values of  $\underline{\alpha}_1$  in the final level PCP  $[\underline{\alpha}_1; \underline{\alpha}_n]$ . Note, since  $\underline{\alpha}_1$  is not the key in any of the PCP's, a cross-product has to be obtained for all the nonkey values  $\underline{\alpha}_2$  to  $\underline{\alpha}_n$  for every  $\underline{\alpha}_1$  value in  $[\underline{\alpha}_1; \underline{\alpha}_n]$ . The EWCR's are taken one at a time from  $[\underline{\alpha}_1; \underline{\alpha}_n]$  and the cross product is formed. In the tuples so obtained the  $\underline{\alpha}_i$  values are replaced by the attributes which have to be output (one attribute from each level). Clearly, this operation will repeat for  $p_n$  times, where  $p_n$  is the number of qualified  $(\underline{\alpha}_1 - \underline{\alpha}_n)$  pair values in  $[\underline{\alpha}_1; \underline{\alpha}_n]$ .

The time  $t_o$  to perform this operation is given by,

$$t_o = p_n (t_n + t_{cp}) + t_{tp} \quad (6.28)$$

where  $t_{cp}$  is the time to perform cross product

$t_n$  is the time to read the value pairs  $\underline{\alpha}_1 - \underline{\alpha}_i$

(for  $i = 1, 2, \dots, n$ ) from the cell I/O buffers

and  $t_{rp}$  is the time to replace the  $\underline{a}_i$  values with the attribute values

$t_{cp}$  and  $t_n$  are very small compared to the revolution time of the track. (Note  $t_{cp}$  is nothing but the instruction time of WCRML instruction JOIN).  $t_{rp}$  would be the time to read the  $n$  I/O buffers from the cells involved and accessing and replacing each of the  $\underline{a}_i$  values in the sliced buffer of the QP. For a large number of these values, this time would be of the order of the revolution time. Finally, the total time for WCRC,  $t_{t2}$  would be

$$\begin{aligned} t_{t2} &= 1 + (3+L)(n-1) + p_n(t_n + t_{cp}) + t_{rp} \\ &= 1 + (3+L)(n-1) + p(t_n + t_{cp}) + t_{rp} \end{aligned} \quad (6.29)$$

(assuming  $p_n = p$ )

Comparing Equation (6.29) with Equation (6.27), it is easy to see that  $t_{t1} \gg t_{t2}$ . However, it must be pointed out that the gain in computation time in WCRC is partly due to the specialized extra hardware such as boolean evaluator and the join and sort unit, at the internal level.

Further, if more than one attribute has to be selected for output from each level, the time  $t_{t2}$  for WCRL would still remain the same because, this would mean more slices of the buffer being selected in the QP. In the case of GDBMS, all the other timings would remain the same except the time for projection,  $t_p$ , which would increase by

a factor  $\frac{N_{dp}}{3}$  where  $N_{dp}$  is the number of domains to be projected on.

Note RAP can project a maximum of 3 domains at a time. [OZKARAHAN 76].

If a  $\theta$ -join has to be performed instead of natural join, the expressions for times  $t_{t1}$  and  $t_{t2}$  still remain the same because RAP can perform  $\theta$ -join as easily as natural join and WCRC would have to perform cross product taking one qualified EWCR at a time, which is already shown above.

#### 6.5 Update Time

For a general update on all domains on one relation  $R_i$  (say), GDBMS would take  $m_i$  revolution where  $m_i$  is the number of domains in  $R_i$ . WCRC would take only 1 revolution with option II, because all update can be done in parallel on the PCP's. WCRC with option I would take  $3 * n_t$  revolutions, where  $n_t$  is the number of tracks which are full following the track in which the update has to be performed. Note option I maintains ordering on the values in a PCP track. Insertion or deletion of a tuple in  $R_i$ , would take one revolution in GDBMS while it takes  $(1\frac{1}{2} * n_t)$  revolutions in WCRC - option I, and 1 revolution in option II.

#### 6.6 Overall Comparison of GDBMS and WCRC

An overall comparison between GDBMS in [DOGAC 80] and WCRC is presented here.

- 1) GDBMS uses RAP as its internal level. The storage of data is n-ary

relations. This limits efficient utilization of data in data models other than relational and also could restrict query optimization. WCRC uses binary PCP's based on WCR's as the storage structure, which avoids these problems. It further avoids any bias between forward and the reverse PCP's by the design of the data storage schemes (option I and II).

ii) GDBMS performance would depend on the query statistics - faster for queries in relational data bases. WCRC is independent of query statistics by virtue of its binary PCP's.

iii) In GDBMS the user views are translated directly into the internal level RAP primitives without going through the conceptual level. This severely limits physical data independence. In WCRC the conceptual level separates the external and internal levels giving better physical data independence.

iv) In GDBMS, the DBA has no DML at the conceptual level, while WCRC supports DBA fully.

v) GDBMS is an SIMD scheme while WCRC is a MIMD schema.

vi) The space requirements for PCP data structures compared to n-ary relations is lesser.

vii) The response time, due to internal level for most of the queries is better for WCRC.

## 6.7 Summary


In this chapter, WCRC is evaluated in comparison with GDBMS and DeFiore's scheme in terms of storage and retrieval. The overall times for retrieval in GDBMS and WCRC are derived and compared. In the next chapter, the concluding remarks and suggestions for future research are presented.

## CHAPTER VII

### DISCUSSION AND FUTURE RESEARCH

#### 7.1 Discussion

This thesis has concentrated on the architectural features of a new data base machine called WCRC. The architecture is well within the ANSI/SPARC proposals. It consists of three levels: external, conceptual and internal levels. It can simultaneously support the three major data models: network, hierarchical and relational at the external level. The overall architecture and the facilities to the user as well as the DBA, have been described. A basis for the conceptual level design is developed. At the conceptual level, the choice of the data model (E-R model) and the data language (WCRL) provide the required stable view of the data base to users at the external level, as well as physical and logical data independence. The algorithms for schema conversion and the view translation are developed. The schema conversion algorithms provide the DBA with the external schemas in different models, equivalent to the conceptual schema in E-R model. These schemas would form an informal basis for defining the user views. Once the user views are defined, they are translated into E-R views and subjected to consistency checking in the conceptual processor. A frame-



work for consistency checking and updates based on the conceptual model, is presented. However, further work is needed for a sophisticated update policy. Some preliminary work on query optimization based on WCRL operations is also reported.

The conceptual level language, WCRL, is further extended to accommodate data definition, data manipulation and storage definition. A specialized high level language, DBAL is developed for the convenience of DBA, which also includes retrieval, data definition and storage definition facilities. The full BNF syntaxes are presented for facilitating translation of these languages in future. The translation of external level languages SEQUEL, LSL, IMS data sublanguage and DBAL into WCRL is described with examples.

At the internal level, the data is stored as PCP's. This is a radical departure from the conventional approaches where data has been stored as n-ary relations. Also, this storage schema provides physical data independence and allows for easier conversion into any of the data models at the external level. Two storage structures (option I and II) are suggested for the storage of PCP's on the tracks. The relative merits and demerits are discussed. On the whole, the PCP option II seems to provide a better storage structure. A machine-oriented WCRML, which is directly executable in hardware, is developed. The basic hardware design to implement these instructions is outlined. Special hardware units for supporting some data base functions such as physical join and sort are developed. So far in the literature, these functions have been mostly supported by soft-



ware only. The extra hardware, where applicable, to implement both options is reported. The guidelines for a detailed design of cell control unit and the query processor, are provided.

Finally, a comparative analysis of WCRC for both options is attempted. The storage requirements and retrieval times are also compared with the GDBMS and DeFiore's scheme. A detailed comparison of WCRC and GDBMS is provided for a general retrieval (including I/O time) and a general update. The results indicate that in a general query the WCRC offers an order of magnitude better performance over GDBMS.

## 7.2 Future Research

The architecture of WCRC reported here opens up several new areas of research. The following lists some major areas that need further investigation and experimentation:

- i) Translation of user languages in Relational, Network, and Hierarchical data bases into conceptual language, WCRL.
- ii) Translation of DBAL into WCRL
- iii) Translation of WCRL to WCRML, the machine level language
- iv) More sophisticated update, concurrency, integrity and security policies.
- v) Implementation and experimentation with the hardware design of cells and some units of the query processors.
- vi) Implementation of conceptual processor and query processor controller on mini/microcomputers.

vii) Extension of WCRC architecture to handle null values and corresponding extensions of the external level languages and the data models.

It may be noted that the research reported in this thesis is just the beginning of a new era in data base computer architecture and system design. The focus of the work reported here has been to provide a well-defined framework for a complete design of a new generation of data base machines.

## REFERENCES

[ANSI 75]

ANSI/X3/SPARC "Interim Report ANSI/X3/SPARC Study Group on Data Base Management Systems," FDT 7 (2), Vol. 7, No. 2, 1975.

[ARORA 79]

Arora, S. K., Smith, K. C., "A Theory of Well-Connected Relations", J. of Information Sciences, 19, 1979, pp. 97-134.

[ARORA 80]

Arora, S. K., Smith, K. C., "WCRL: A Data Model Independent Language for Data Base Systems", To appear, Int. J. of Comp. and Inf. Sciences, 1980.

[ARORA 80A]

Arora, A. K., Carlson, C. R., "On the Updatability of Consistent Relational Views", Technical Report, Bell Labs, Naperville, Illinois, 1980.

[ARORA 81]

Arora, S. K., Dumpala, S. R., Smith, K. C., "WCRC: An ANSI SPARC Machine Architecture for Data Base Management", To appear, Proc. International Symposium on Computer Architecture, Minneapolis, May 12-14, 1981.

[BABB 79]

Babb, E., "Implementing a Relational Data Base by Means of Specialized Hardware", ACM Trans. on Data Base Systems", Vol. 4, No. 1, Mar. 1979.

[BACHMAN 69]

Bachman, C. W., "Data Structure Diagrams", Data Base 1 (2), 1969, pp. 4-10.

[BANERJEE 79]

Banerjee, J., Hsiao, D. K., Kannan, K., "DBC - A Data Base Computer for Very Large Data Bases", IEEE Trans. on Computers, Vol. C-28, No. 6, June 1979, pp. 414-429.

## [BOYCE 74]

Boyce, R. F., Chamberlin, D. D., King, W. F., III, and Hammer, M. M. [1974]. "Specifying Queries as Relational Expressions", in Data Base Management (Klimbie, J. W., and Koffeman, K. L., eds.), North-Holland, Amsterdam, pp. 169-176.

## [BOYCE 75]

Boyce, R. F., Chamberlin, D. D., King, W. F., and Hammer, M. M., "Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage", Comm. ACM 18:11, 1975, pp. 621-628.

## [BRACCHI 74]

Bracchi, G., Fedeli, A., and Paolini, P., "A Multilevel Relational Model for Data Base Management Systems", in Data Base Management (Klimbie, J. W., and Koffeman, K. L., eds), North-Holland, Amsterdam, 1974, pp. 211-223.

## [BUNEMAN 79]

Buneman, P., Frankel, R. E., "FQL - A Functional Query Language", ACM-SIGMOD, Boston, 1979, pp. 52-58.

## [CHAMBERLIN 74]

Chamberlin, D., Boyce, R., "SEQUEL: A Structured English Query Language", ACM-SIGMOD Workshop on Data Description, Access and Control, May 1974, pp. 249-264.

## [CHAMBERLIN 75]

Chamberlin, D. D., Gray, J. N., and Traiger, I. L., "Views, Authorization and Locking in a Relational Data Base System", Proc. AFIPS 44, NCC, 1975, pp. 425-430.

## [CHAMBERLIN 76]

Chamberlin, D. D., et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control", IBM J. Res. 20:6, 1976, pp. 560-575.

## [CHEN 76]

Chen, P. P. S., "The Entity Relationship Model - Towards a Unified View of Data", ACM Trans. on Data Base Systems, March, 1976, Vol, 1, No. 1, pp. 9-36.

## [CODASYL 71]

CODASYL DBTG, CODASYL Data Base Task Group Report, Conf. Data Sys. Languages, ACM, New York, 1971.

## [CODD 70]

Codd, E. F., "A Relational Model of Data For Large Shared Data Banks", CACM, 1970, pp. 377-387.

## [CODD 79]

Codd, E. F., "Extending the Data Base Relational Model to Capture More Meaning", ACM Trans. Data Base Systems, December 1979, Vol. 4, pp. 397-434.

## [COPELAND 74]

Copeland, G. P., "A Cellular System for Non-Numeric Processing", Ph.D. Dissertation, Department of Electrical Engineering, University of Florida, 1974.

## [DALE 76]

Dale, A. G., Dale, N. B., "Schema and Occurrence Structure", ACM SIGMOD, Washington, D.C., 1976, pp. 157-168.

## [DALE 77]

Dale, A. G., Dale, N. B., "Main Schema - External Schema Interaction in Hierarchically Organized Data Bases", ACM SIGMOD, Toronto, 1977, pp. 102-110.

## [DATE 76]

Date, C. J., "An Introduction to Data Base Systems", First Edition, Addison-Wesley, 1976.

## [DeBLASIS 78]

DeBlasis, J. P., Johnson, T. H., "Review of Data Base Administrators Functions from a Survey", ACM SIGMOD, Austin, 1978, pp. 101-109.

## [DeFIORE 74]

DeFiore, C. R., and Berra, P. B., "A Quantitative Analysis of the Utilization of Associative Memories in Data Management", IEEE Trans. Computers, Vol. C-23, No. 2, 1974, pp. 121-132.

## [DeWITT 78]

DeWitt, D. J., "DIRECT - A Multiprocessor Organization for Supporting Relational Data Base Management Systems", Proc. Fifth Annual Symp. on Comp. Architecture, 1978, pp. 182-189.

## [DOGAC 80]

Dogac, A., Ozkarahan, E. A., "A Generalized DBMS Implementation on a Data Base Machine", ACM SIGMOD, Santa Monica, California, May 1980, pp. 133-143.

## [FOSTER 76]

Foster, C. C., "Content Addressable Parallel Processors", Reinhold Co., 1976.

## [FURTADO 79]

Furtado, A. L., Sevcik, K. C., dos Santos, C. S., "Permitting Updates Through Views of Data Bases", Information Systems, 1979, Vol. 4, No. 4, pp. 269-284.

## [HOLLAR 78]

Hollar, L. A., "A Design for a List Merging Network", IEEE Trans. on Computers, June 1979, Vol. C-28, No. 6, pp. 406-413.

## [HOUSEL 79]

Housel, B. C., "QUEST: A High Level Data Manipulation Language for Network, Hierarchical and Relational Data Bases", IBM Res. Rep., RJ2588 (33488) 7/25/79, 1979.

## [IBM 75]

IBM, Information Management System/Virtual Storage (IMS/VS) Publications; Application Programming Reference Manual, SH20-9026-2; System Programming Reference Manual, SH20-9027-2; IBM Corp., White Plains, New York, 1975.

## [KLUG 78]

Klug, A., Tsichritzis, D., "Multiple View Support Within ANSI/SPARC Framework"; VLDB, Tokyo, 1978, pp. 477-488.

## [KNUTH 73]

Knuth, D. E., "The Art of Computer Programming 3, Sorting and Searching", Addison-Wesley, 1973.

## [LEILICH 78]

Leilich, H. O., Stiege, G., Zeidler, H. Ch., "A Search Processor for Data Base Management Systems", Proc. Fourth VLDB, West Berlin, Sept. 1978, pp. 280-287.

## [LEWIS 78]

Lewis, E. A., Sekino, L. C., Ting, P. D., "Data Semantics and Data Base Update Rules Based on Functional Dependencies", Bell Labs., Technical Report, Holmdel, N.J., 1978.

## [LIN 76]

Lin, C. S., Smith, D. C. P., Smith, J. M., "The Design of a Rotating Associative Memory for Relational Data Base Applications", ACM TODS, Vol. 1, No. 1, March 1976, pp. 53-65.

[LIPOVSKI 78]

Lipovski, G. J., "Architectural Features of CASSM: A Context Addressed Segment Sequential Memory", Proc. 5th Annual Symposium on Computer Architecture, Palo Alto, Calif., April 1978, pp. 31-38.

[McGREGOR 76]

McGregor, D. R., Thomson, R. G., and Dawson, W. N., "High Performance for Data Base Systems", Systems for Large Data Bases, North-Holland Publishing Co., 1976, pp. 103-116,

[MILLER 65]

Miller, R. E., "Switching Theory", Wiley, 1965.

[NIJSSSEN 76]

Nijssen, G. M., "A Gross Architecture for the Next Generation Data Base Management Systems", Modelling in Data Base Management Systems, North-Holland, 1976, pp. 1-24.

[OLIVER 79]

Oliver, E. J., "RELACS, An Associative Computer Architecture to Support a Relational Data Model", Ph.D. Thesis, Syracuse University, 1979.

[OSMAN 79]

Osman, I. M., "Updating Defined Relations", National Computer Conference, 1979, pp. 733-740.

[OZKARAHAN 75]

Ozkarahan, E. A., "An Associative Processor for Relational Data Bases - RAP", Ph.D. Thesis, Dept. of Computer Science, University of Toronto, 1975.

[OZKARAHAN 76]

Ozkarahan, E. A., Schuster, S. A., "A High-Level Machine Oriented Assembler Language for a Data Base Machine", Technical Report, CSRG-74, October, 1976.

[PARHAMI 73]

Parhami, B., "Associative Memories and Processors: An Overview and Selected Bibliography", Proc. IEEE, Vol. 61, No. 6, June 1973, pp. 727-730.

[PARKER 71]

Parker, J. L., "A Logic Per Track Device", Proc. IFIP Cong. 1971, North-Holland Pub. Co., Amsterdam, pp. TA4-146-TA4-150.

## [SCHMID 75]

Schmid, M. A., and Swenson, J. R., "On the Semantics of the Relational Data Model, ACM SIGMOD International Conference on Management of Data, San Jose, California, 1975, pp. 211-223.

## [SCHUSTER 79]

Schuster, S. A., Nguyen, H. B., Ozkarahan, E. A., Smith, K. C., "RAP 2 - An Associative Processor for Data Bases and its Applications", IEEE Trans. on Computers, June 1979, Vol. C-28, No. 6, pp. 446-458.

## [SENKO 73]

Senko, M. B., Altman, E. B., Astrahan, M. M., and Fehder, P. L., "Data Structures and Accessing in Data Base Systems", IBM Sys. J. 12, 1973, pp. 30-93.

## [SU 79]

Su, S. Y. W., Nguyen, L. H., Emam, A., Lipovski, G. J., "The Architectural Features and Implementation Techniques of the Multi-cell CASSM", IEEE Trans. on Computers, June 1979, Vol. C-28, No. 6, pp. 430-445.

## [SU 79A]

Su, S. Y. W., "Cellular-Logic Devices: Concepts and Applications", Computer, March 1979, pp. 11-25.

## [TSICHRITZIS 76]

Tsichritzis, D., "LSL: A Link and Selector Language", ACM-SIGMOD, Washington, D.C., June 1976, pp. 123-134.

## [TSICHRITZIS 77]

Tsichritzis, D. C., Lochovsky, F. H., "Data Base Management Systems", Academic Press, 1977.

## [ULLMAN 80]

Ullman, J. D., "Principles of Data Base Systems", Computer Science Press, 1980.

## [VASSILIOU 80]

Vassiliou, Y., "Functional Dependencies and Incomplete Information", Proc. VLDB, Montreal, 1980, pp. 260-269.



## APPENDIX A

### THE BNF SYNTAX OF WCRL

#### A.1 Notation

This Appendix defines the syntax of WCRL using Backus-Naur form productions. The productions have been simplified for the sake of illustration. For example, the term "EWCR List" is used to denote " $w_1 [A_1; B_1], w_2 [A_2, B_2], \dots, w_n [A_n; B_n]$ ".

#### A.2 Syntax

Statement := Retrieval statement

- | DDL Statement (E-R Model)
- | Update Statement
- | SDL Statement.

Retrieval statement := Set Reconstitution Statement

- | set Join
- | Selective Union
- | Selective Intersection
- | Selective Difference
- | Recursion Statement



$\langle \text{name list} \rangle := \langle \text{name type} \rangle , \langle \text{name list} \rangle$   
 $\quad | \langle \text{name type} \rangle$

$\langle \text{conditional expression} \rangle := \langle \text{cond clause} \rangle \langle \text{bop} \rangle \langle \text{conditional expression} \rangle$   
 $\quad | \langle \text{cond clause} \rangle$

$\langle \text{cond clause} \rangle := \langle \text{Attribute name} \rangle \langle \text{cop} \rangle \langle \text{opd} \rangle$   
 $\quad | \langle \text{oprd} \rangle \langle \text{sop} \rangle \langle \text{oprd} \rangle$

$\langle \text{bop} \rangle := \wedge | \vee | \neg$

$\langle \text{cop} \rangle := = | \neq | > | < | \geq | \leq$

$\langle \text{sop} \rangle := \supset | \subset | \equiv | \neq | \subseteq | \supseteq | \emptyset | \neq \emptyset$

$\langle \text{opd} \rangle := \langle \text{Attribute name} \rangle | \text{constant}$

$\langle \text{oprd} \rangle := \langle \text{func} \rangle \langle \text{Attribute name} \rangle$

$\langle \text{func} \rangle := \langle \text{aggr func} \rangle | \langle \text{set func} \rangle$

$\langle \text{set func} \rangle := \text{some} | \text{current} | \text{all}$

Selective Union : =  $\sigma_{su} \langle \text{name list} \rangle (\langle \text{conditional expression} \rangle)$

Selective Intersection : =  $\sigma_{si} \langle \text{name list} \rangle (\langle \text{conditional expression} \rangle)$

Selective Difference : =  $\sigma_{sd} \langle \text{name list} \rangle (\langle \text{conditional expression} \rangle)$

Recursion Statement : =  $\sigma_{\langle \text{id} \rangle} \langle \text{cp name type} \rangle \langle \text{eq op} \rangle \sigma_{\langle \text{id} \rangle}$   
 $\quad \langle \text{Retrieval statement} \rangle$

$\langle \text{id} \rangle := \langle \text{character string} \rangle$

DDL Statement : = Entity definition statement  
 $\quad |$  Relationship definition statement  
 $\quad |$  Attribute definition statement  
 $\quad |$  Constraint specification statement  
 $\quad |$  Schema change specification statement

Entity definition statement :=  $\sigma_{de}$  <attribute list> (<E\_specification>)

<eq op> e(<ename>)

<E\_specification>:= (<etype>, <attribute name>, <ename>)

<ename>:= <character string>

<etype>:= normal|weak

<rname>:= <character string>

Relationship definition statement:=  $\sigma_{dr}$  <attribute list> (<R\_specifi-

cation>) <eq op> r (<rname>)

<R\_specification>:= (<deg>, <spec list>)

<deg>:= Binary|Ternary|Tertiary|k-ary

<spec list>:= <spec>, <spec list>

|<spec>

<spec>:= (<ename>, <ename>, <rtype>)

<rtype>:=  $\tilde{1}:1|\tilde{1}:N|\tilde{N}:M|1:\tilde{1}|1:\tilde{N}|\tilde{N}:M|1:1|1:N|N:M$

Attribute definition statement :=  $\sigma_{da}$  <vname> (<V\_specification>)

<eq op> a(<aname>)

<vname>:= <character string>

<V\_specification>:= (<data type>, <unit>, <base type>)

<data type>:= <Integer>|<real>|<character string>

<unit>:= <character string>

<base type>:= <scalar type>|<subrange type>

<scalar type>:= <constant>, <scalar type>|<constant>

<subrange type>:= <constant> - <constant>

<aname>:= <character string>|<ename>.<character string>

Constraint specification statement: = security statement  
 | integrity statement

Security statement: =  $\sigma_{sc}$  <object list> (<sexp list>)

<object list>:= <attribute list>|<entity list>

<entity list>:= <ename>|<ename>, <entity list>

<sexp list>:= <sexp>|<sexp>, <sexp list>

<sexp>:= (<user no>, (<access list>))

<user no>:= Integer

<access list>:= <access type> <access type>, <access list>

<access type>:= R|W|M|S

Integrity statement: =  $\sigma_{IC}$  <object name list> (<iexp list>)

<object name list>:= <object name>|<object name>, <object name list>

<iexp list>:= <iexp>|<iexp>, <iexp list>

<iexp>:= <aggr func> <object name> <oper> <opd>

<object name>:= <ename>|<rname>|<aname>|<vname>

<oper>:= <cop>|<sop>

<operand>:= <constant> <aggr func> (<object name>)

Schema change specification statement:=  $\sigma_{cs}$  <object spec list>

[<object type list>] (<condition list>) <eq op> o(<object name list>)

<object spec list>:= <object name>; <object spec list>|<object name>

<object type list>:= <object type>|<object type>; <object type list>

<object type>:= entity|relationship|attribute|value set

<condition list>:= <condition>; <condition list>|<condition>

<condition>:= <object name> <cop> <constant>|<>null>

Update Statement: = insertion

|deletion

|modification

insertion: =  $\sigma_I$  <attribute list> [<type name>] (<i expression list>)  
 <eq op> o(<type name>)

<type name>: = <ename> | <rname>

<iexpression list>: = <iexpression>; <iexpression list> | <iexpression>

<iexpression>: = <value list> | <iexpr list>

<value list>: = <constant>, <value list> | <constant>

<iexpr list>: = <iexpr>, <iexpr list> | <iexpr>

iexpr: = aggr func (<aname>)

|aggr func (<aname>).<op> <constant>

deletion: =  $\sigma_D$  <attribute list> [<type name>] (<yexpression list>)  
 eq op o(<type name>)

<yexpression list>: = <yexpression>; <yexpression list> <yexpression>

<yexpression>: = <value list> <yexpr list>

<yexpr list>: = <yexpr>, <yexpr list> | <rexpr>

<yexpr>: = <aggr func> <aname> <op> <constant>

|<aname> <cop> <aggr func> <aname>

modification: =  $\sigma_M$  <attribute list> [<type name>] (<yexpression list>)  
 (<value list>) <eq op> o (<type name>)

SDL statement: = mapping statement

; | storage definition statement.



## APPENDIX B

### THE BNF SYNTAX OF DBAL

This Appendix defines the syntax of Data Base Administrator Language (DBAL) using Backus-Naur Form productions.

```
DBAL Command := DDL command  
              | DML command  
              | SDL command
```

```
DDL command := DEFINE <block>  
             | GRANT <grant block>
```

```
<block> := <value set block>  
         | <attribute block>  
         | <entity block>  
         | <relationship block>  
         | <weak entity block>  
         | <constraint block>
```

```
<value set block> := VALUE_SET <value set name> OF  
                   DATA TYPE <data type (<number>)> WITH  
                   PREDICATE (<base type>, <unit type>)
```

```
<value set name> := <character string>
```

```
<data type> := <integer> | <real> | <character string>
```

```
<number> := <integer>
```





<weak entity block>:= WEAK\_ENTITY\_SET <entity name> WITH  
 ATTRIBUTES <attribute list>  
 KEY <attribute name>  
 RELATIONSHIP <relationship name>

<constraint block>:= ON <object name>  
 CONSTRAINT <expression>

<object name>:= <entity name>  
 |<relationship name>  
 |<value set name>  
 |<attribute name>

<expression>:= <object name> <opr> <opd>

<opr> := = | ≠ | < | > | ≤ | ≥ | < | > | > | > | = | ≠

<opd>:= <aggr func> <object name>  
 |<constant>

<aggr func>:= SUM|AVG|MIN|MAX|CAR|ALL|SOME.

<grant block>:= TO (<user number list>)  
 ON <object name>

<user number list>:= <user number>, <user number list>  
 <user number>

<user number>:= <integer>

DML command := <add command>  
 |<delete command>  
 |<split command>  
 |<merge command>  
 |<shift command>

|<rename command>  
 |<insert command>  
 |<delete command>  
 |<modify command>  
 |<retrieve command>

<add command>:= ADD <object name> TO <object type>

<object type>:= Entities|Relationships|value sets|attributes

<delete command>:= DELETE <object name> FROM <object type>

<split command>:= SPLIT <obj type> <object name>

INTO <object name list>

QUALIFICATION <expression>

<object name list>:= <object name> , <object name>

<merge command>:= MERGE <object type> <object name list>

INTO <object name>

<shift command>:= SHIFT (<mode>) <object name>

TO <object name>

<mode>:= high|low

<rename command>:= RENAME <object name>

TO <object name>

<insert command>:= INSERT INTO <object name> <tuple block>

<tuple block>:= <tuple>, <tuple block>|<tuple>

<delete command>:= DELETE FROM <object name>

(<tuple>|WHERE <condition>)

<condition>:= <expression>

<modify command>:= MODIFY <object name>

(<tuple>|WHERE <condition>)

```

<retrieve command>:= RETRIEVE <argument list>
                        WHERE <cond expr>
<argument list>:= <item>, <argument list>
                  |<item>
<item>:= <term> USING <navigation>
        |<term>
<term>:= <entity name>
        |<entity name>.(<attribute list>)
        |<aggr func> (<entity name>).( <attr list>)
        |<relationship name>
        |<relationship name> (<attribute list>)
        |<aggr func> (<entity name>).( <attr list>)
<attr list>:= <attr>, <attr list> <attr>
<attr>:= <attribute name> [<free variable>]
<free variable>:= <character>
<navigation>:= <path> , <navigation>
              |<path>
<path>:= <entity name> , <relationship name> , <entity name>
        |<relationship name> , <entity name>, <relationship name>
<cond expr>:= <expr> <bop> <cond expr>
<expr>:= <aggr func> <attribute name> <opr> <opd>
<bop>:= AND|OR
SDL command:= <CP command>
             |<ER command>
             |<RR command>

```

<CP command>: = DECLARE\_CP <cp name>  
ON (<attribute name> , <attribute name>)

<cp name>: = <character string>

<ER command>: = DECLARE\_ER <entity name>  
AS <cp name list> KEY <attribute name>

<RR command>: = DECLARE\_RR <relationship name>  
ON <cp name list> KEY <attribute name>

<cp name list>: = <cp name> , <cp name list>  
<cp name>

```

L2  RESET (t1t2t3) [R]      /*Reset all mark bits of tuples
                             just t3 marked*/ (1 revolution)
L3  TEST t1_RAIL           /*Any more tuples to be processed?*/
    BC L1,RAIL_STAT(t1)    /*Yes, repeat the loop*/
                             /*All tuples with unique Di values are
                             left t2 marked. They can be left for
                             future processing or read out as the
                             following code shows*/

    READ[R:MKED(t2)] [WORK AREA]
    EOQ.

```

The second part of this appendix shows the time taken for a physical join of two relations R and S over domain  $D_1$ . The corresponding RAP program is shown below. The preprocessing takes 1 revolution and the loop  $L_1$  loops around for p times, where p is the number of marked tuples in R. The inner loop (RD LOOP), loops around for q times, where q is the number of tuples that qualify for join condition in s for every tuple in p. Clearly, the outer loop needs 5 revolutions and the inner loop requires 4 revolutions. The re-setting of  $R_1$  takes 1 revolution.

$$\text{The total time} = 5p + 4pq + 2$$

Note, if more than two relations are involved in the join, then, the instructions marked '\*' in the RAP program will have to be repeated for (n-1) times, where n is the total number of relations.

i.e., 3 out of 5 instructions in the outer loop would be repeated  
(n-1) times for each selected tuple from R.

$$\text{The total time} = 2p + 4p \sum_{i=1}^{n-1} q_i + 3p(n-1) + 2$$

The RAP program for JOIN of two relations R and S

```

MARK (t1) [R] /*Preprocess for loop*/ (1 revolution)
/*Loop through tuples of relation R*/
L1 GET_FIRST [R(D1):MKED(t1)] /*Read in the domain value of
the current tuple*/ (1 revolution)
*MARK(t2)[S:D1 0 (REGC_1)] /*Mark all tuples of S with domain
values satisfying the join condition*/
(1 revolution)
TEST t2_RAIL /*Any such S tuple?*/
BC L2, RAIL_STAT(t2) /*Yes*/
BC SKIP /*No*/
/*Now read out alternately, the current
R tuple and one of the t2 marked S
tuple until all t2 marked S tuples
have been processed*/
*L2 MARK (t3) [S:UNMKED(t2)] /*Preprocess for loop*/ (1 revolution)
RDLOOP READ [R:UNMKED(t1t4)] /*Read the R tuple*/ (1 revolution)
[WORK AREA]
GET_FIRST[S(D1):MKED(t2)] /*t2 unmark the first t2 marked S tuple
(The D1 value saved is not used)*/
(1 revolution)

```

```

READ[S:UNMKED(t2t3)] [WORKAREA]/*Read out the S tuple just t2 unmarked*/
                                                    (1 revolution)
MARK (t3) [S:UNMKED(t2t3)] /*Mark it as processed*/ (1 revolution)
TEST t2_RAIL /*Any more S tuples to be processed?*/
                                                    (1 revolution)
BC RDLOOP,RAIL_STAT(t2) /*Yes, repeat the loop*/
*RESET (t3) [S] /*No, clear work mark bit*/
                                                    (1 revolution)
/*Housekeeping before we can proceed to
the next tuple of the relation R*/
SKIP MARK(t4)[R:UNMKED(t1t4)] /*Mark tuple as processed*/
                                                    (1 revolution)
TEST t1_RAIL /*Any more R tuples to be processed?*/
BC L1,RAIL_STAT(t1) /*Yes, repeat the loop*/
/*All tuples of relation R have been
processed. Clean up and exit*/
RESET [t4] [R] (1 revolution)
EOQ

```