RAP 2
AN ASSOCIATIVE PROCESSOR FOR DATA BASES

S. A. SCHUSTER
H. B. NGUYEN
E. A. OZKARAHAN
K. C. SMITH

FINAL REPORT TO DOC - CONTRACT 0SU5-0205

UNIVERSITY OF TORONTO
JANUARY 1978

Queen
P
91
C655
R36
1978

(1)

RAP 2:
an association processors for data bases:
final report to DOC/

MAR - 8 1978
135

DD 3954436
DL 3965642

P
91
C655
R36
1978

TABLE OF CONTENTS

0. OVERVIEW

0.0.0. Overview

This report is intended to summarize, in a modest way, a jor effort conducted at the University of Toronto by the Computer Systems Research Group with staff and students of the Department of Computer Science, the Department of Electrical Engineering and staff and resources of the Computer Research Facility (CRF). The project received direct support of $199,065 via the Department of Supply and Services (DSS) and the Department of Communications (DOC) of the Government of Canada. In addition it has benefitted by close relationships with a variety of individuals and organizations who have provided a great deal of assistance both motivational and tnagible. Very high on the long list of individuals and organizations are, Dr. A.R. Elliott of the Department of Communications, Dr. R.F. Webb of R.F. Webb Corporation LTd. and Gordon Moore of Intel Corporation.

Direct costs of the RAP project from all sources including those supported by National Research Council (NRC) operating grant funds are estimated to be $218,000. Not included are expenditures by DOC on market studies through R.F. Webb Corporation Ltd. and by INTEL on various associated studies (see Section 9.2).

The RAP project itself occupied an elapsed time of 23 months following the original 15 month period of definition by E.A. Ozkarahan in preparing his Ph.D. thesis entitled "An Associative Processor for Relational Data Bases RAP", presented in Jaunary 1976. A total of 21 persons worked on the project at the University of Toronto praviding a total of more than 12 man years since 1975 and 1 1/2 (largely by EA) before than time.

The summary presented here is a very brief distillaion of a large amount of documented concept, design, experiment, and interacion with others. Estimates of its statistics include:

a) A 350 page thesis.
b) 6 CSRG reports of 225 pages in total.
c) A categorized documentation system estimated to hold 1,500 pages. The cataglouging system and table of contents is included in Section 9.1 for interest.
d) 8 conference/workshop presentations.

e) 6 papers published in journals/proceedings, of 60 pages in total.
f) Visits by RAP staff to 18 organizations as part of an industrial liason program leading to the study reported by R.F. Webb.
g) Visits by 30 individuals from 10 organizations to view the RAP work.
h) 75 written enquiries and requests for RAP information.
i) 15 interim/progress reports to DOC.
j) 12 terminal demonstrations of RAP locally.
k) 1 terminal demonstration of RAP elsewhere.
l) 3 M.Sc. completed.
m) 1 Ph.D. completed.

The present status of RAP is a physical system demonstrable within the boards of timescale and availability characteristic of a multiple user system of the computer-hardware consisting of:

1) One communications-linked input terminal running SEQUEL or RAP connected to a:
2) PDP 11/45 well-equipped processor running UNIX, connected to a:
3) PDP 11/10 controller via a 100 foot, 3 byte-paralle link, driving a:
4) CELLBUS with 250 cell capacity to which is attached:
5) Two CELLS each having:
6) CCD Memory of 320K bits capacity, all controllable also from
7) an input terminal to the PDP 11/10 controller communicating in RAP ASSEMBLER.

Read on!

1. INTRODUCTION

RAP - a Relational Associative Processor - is a back-end or peripheral device designed to augment a general purpose computer for implementing a data base management system (DBMS). Its architecture is based on the fact that data base operations are inherently set-oriented and data base addressing is best accomplished through associative reference to achieve high data independence. RAP utilizes these characteristics by combining the features of associative and array processors.

Previous publications on RAP have dealt separately with the details of the first version of its architecture (1,2,3,4), language interface (5,6,), and performance evaluation (7,8,9,). Also, these topics were discussed strictly from the point of view of the relational model of data. This report provides details on the recently evolved, faster, and more flexible architecture and discusses RAP's applicability to a wider range of data base models and its various uses and roles within a complete DBMS implementation. Further its role in communications is highlighted.

In this introduction, we will try to motivate the need for a RAP-like device in DBMS implementations. First, we review some important requirements for these systems. A discussion follows of the limitations of conventional computers in meeting these requirements. The philosophy of using array/associative machines for data base management is then presented. The introduction concludes with the global organization of the RAP processor.

1.1 Data Base Management Systems Requirements

Data base management systems attempt to isolate the details of data storage and manipulation from the application programmers that use the data (10). This is accomplished by providing users a logical view of a data base distinct from the details of its physical realization, a query or data manipulation language by which users can specify or program the retrievals and updates of data, and a data definition language to define and control access to data.

The success to which DBMS can realize the above goals depends heavily upon meeting the following three important requirements. First, query languages must be sufficiently high-level and powerful to permit non-procedural specifications of complex data manipulation. Such languages allow users to specify manipulations in a set-oriented fashion (e.g., retrieve or update all employees...) and addresses of the data associatively by qualifying items stored in the data base (e.g.,... employees who work in the furniture department). The higher the level of query language, the faster and easier it is to code understandable and reliable application systems. Also, high-level query languages can easily be extended to provide stand-alone interactive facilities to users, e.g. engineers, physicians, financial analysts, etc., who are not computer specialists but who are technically oriented and specialize in the semantics of the data being processed.

Secondly, fast response is required because the majority of DBMS will be required to operate in on-line and concurrent user environments which support users at terminals, batch application programs running within multi-tasking systems, and communication and networking systems providing access to distributed data bases.

The third requirement concerns the technical administration of data base systems. The separation of the logical view of data from its physical reality, as provided by data base systems, forces the responsibility of providing efficient performance to be delegated to the data base administrator of the system. This responsibility was originally distributed over several users and distinct file processing systems. Now the data base administrator must make decisions, often conflicting with individual user requirements, for all users. The complexity of this task is enormous in current implementations. It is imperative that tuning of data base systems be effectively and easily achieved.

1.2 Problems with Conventional Implementation

Using conventional computers and traditional secondary memories to implement data base systems makes it difficult to realize the requirements of data base systems and contributes greatly to the high cost of using data bases. The problems of conventional computers arise because of their processing and addressing structure. Conventional machines are designed to process data serially (i.e., the execution of one instruction on a single data item at a time) and that accessing of the data to be processed is accomplished by specifying its location or hardware address. Unfortunately, this is in direct conflict with set-oriented processing and associative addressing in data base systems.

The lack of set-oriented and associative features in conventional architectures force the implementors to achieve equivalent operations by simulating them with software. Accordingly, to implement efficient access to a data base, large complex programs must be written to provide <u>mapping mechanisms</u> that map the users view into the structure of the hardware and provide <u>access paths</u> to facilitate fast location of arbitrary portions of the data base. Access paths refer to both indexing data and software designed to provide access to specific items of the data base. Additional software must also be provided to maintain access paths. Some of the problems caused by the use of access paths and mapping mechanisms in conventional computer implementations of DBMS are:

a) Unbalanced Performance:

Access paths provide fast retrieval of data at the expense of slower updates because updates to the data base must also be reflected in the access paths.

b) Poor Generalized Performance:

Generalized systems based on access paths rely on multi-level software do not permit sufficiently tunable generalized systems that respond to widely varying applications in dynamic user environments.

c) Reduced Potential for Exploiting Concurrency:
Because access paths must be updated whenever the data base is modified, they become additional critical resources that require synchronization to avoid interference in a shared environment. The additional synchronization adds overhead to a data base system.

d) Reduced Reliability:
System failures, especially during updates, can cause the access paths and mapping mechanisms to become inconsistent with the data base. Detection and recovery can be very complex and time consuming.

e) Extra Storage Requirements:
Access paths are often implemented by "inverting" the values occurring in the data base. This technique organizes the physical addresses of data items into search data base creating a second data base requiring extra storage.

f) Software Complexity:
As a consequence of added data structures for implementing access paths and mapping techniques, large complex software is required that is unreliable and dificult to maintain and administer.

A major consequence of these problems is that only those queries whose formulation were preconceived can be conveniently and efficiently processed. The result is that general purpose data base management systems, which must meet the requirements of many applications over broad user environments, perform in a limited fashion in practice. The traditional approach to solving the performance problem has been to acquire faster CPUs and larger primary memories at, of course, tremendous expense. Clearly, this solution does not solve any data base administration problems.

1.3 The Associative/Array Processor Approach

A solution to the limitations of conventional computers for implementing and administrating a DBMS would be to establish a computer architecture that eliminates, or, at least, reduces the dependency on complex software. The new architecture should close the gap between the users logical view and processing requirements and the way the data is represented and processed physically. This can be partially accomplished by utilizing a high degree of parallelism for both processing and addressing. If the parallelism is achieved by repeated cellular logic each operating on its own memory, then the data base system requirement of performing the same operation on many data item operands can be efficiently accomplished. Inexpensive associative memories can be achieved by using block addressable serial memories and searching their contents at high speed in parallel.

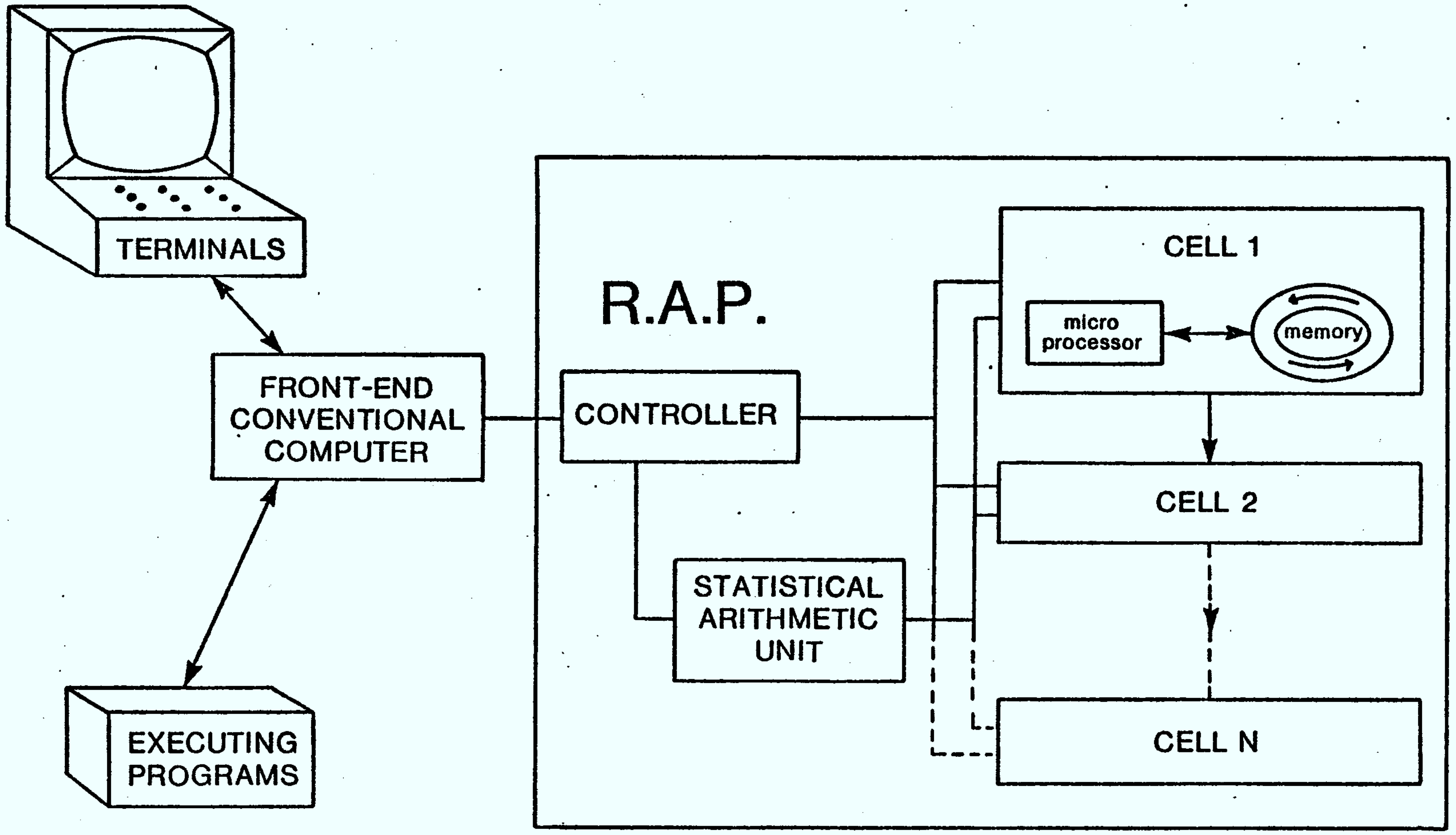
Furthermore, if such device were designed to augment, rather than replace, a conventional computer, it would have a greater chance of overcoming the natural resistanc to a radically new technology by providing a evolutionary approach to total DBMS architecture. Other benefits of moving the majority of data base processing to a back-end organization can be found in (11). We will discuss them with respect to RAP shortly. Designs other than RAP, that prescribe to this approach in various degrees, can also be found in the literature (12-16).

1.4 Basic Architecture of RAP

The basic architecture of a RAP device consists of a "chain" of parallel components called cells, a statistical arithmetic unit, and central controller. This organization is shown in Figure 1. Each cell is composed of a processor and block addressable memory. The processor is specifically constructed for data base definition, insertion, deletion, update, and retrieval primitives. Logic for each processor has been designed to be compatible with LSI circuit implementation technology. The memory can be implemented by a rotating magnetic device such as the track of a disk or drum, semiconductor CCD, or bubble memory. The statistical arithmetic unit is designed for computing summary statistics (e.g. totals, averages, etc. over the combined contents of the cell memories. The controller is responsible for receiving instructions in RAP machine format from a general purpose front-end, computer decoding them, broadcasting control sequences to initiate cell execution, and passingretrieved or inserted items between the front-end and RAP. Each RAP instruction is executed within the cells which operate in parallel directly on the data. Simple intercell communication for priority polling is implemented along the chain. Each memory contains data formated into a sequence of records containing values of data items. The details will be given shortly.

A cell is composed of several logic units. The most important being involved with searching. Several comparitor elements form the basis of the associative addressing architecture of a cell. The comparitors can independently test the contents of one item in the data base against several literals or several items each against different literals. The true or false results of comparison tests on a record can be combined into a disjunctive or conjuctive result to determine if the record associatively qualifies for further manipulation.

The front-end computer supports high-level user functions. It interfaces users to to RAP by supporting communications via interactive terminals or through programming language CALL and I/O statements for application programs running in batch multi-programming operating systems. The translation of various query languages into RAP programs will also be accomplished in the front-end. Data base system software responsib for coordinating multiple and diverse secondary storage devices other than RAP, schedulin of queries, and maintaining protection, security, and integrity must also be supported in the front-end but can be aided by the data processing capabilaties of RAP.


TERMINALS
FRONT-END CONVENTIONAL COMPUTER
EXECUTING PROGRAMS
R.A.P.
CONTROLLER
STATISTICAL ARITHMETIC UNIT
CELL 1
micro processor
memory
CELL 2
CELL N

2. THE ABSTRACT MACHINE

The RAP system has a machine-oriented yet high-level and complete instruction set for manipulating its data base. Most macro-assembler instructions correspond to one machine instruction which involves several cell micro-code instructions. In this section, an explanation of the RAP macro-assembler instructions will be presented. A programmers view of the RAP data structure will be given first. Then the basic structure of a RAP instruction will be given followed by the description of each individual instruction.

2.1 Data Structure

From a programmer's view, RAP stores data as unordered occurrences of records defined by a RAP relation as shown in Figure 2. A relation can be envisioned as a formatted table of data where rows of the table represent a set of record occurrences sometimes called tuples in relational terminology. The occurrences of a relation stores data about a set of similar entities (e.g. persons, places, things, or relationships). The name of a relation identifies the set of entities. The format of record occurrences is defined by naming the data items whose concatenated values occur in each record and specifying their length. The length of each item in the relation is fixed according to a users choice of one of several sizes. Each occurrence of a relation stores data which describes a particular entity by assigning a value to each of the items according to the format of the relation The values are treated internally as simple bit patterns for non-numeric data and as integers in twos - complement format for numeric data.

Each relation and its occurrences are augmented by several special one bit items called mark bits. These items can be set to 0 or 1 under user control through various marking instructions or by the intermediate operations of other instructions. The bits are used primarily as a work area to temporarily indicate subsets of record occurrences so that the results of one instruction can be used in subsequent instructions. This is done by treating the mark bits as normal data items to be tested during associative addressing.

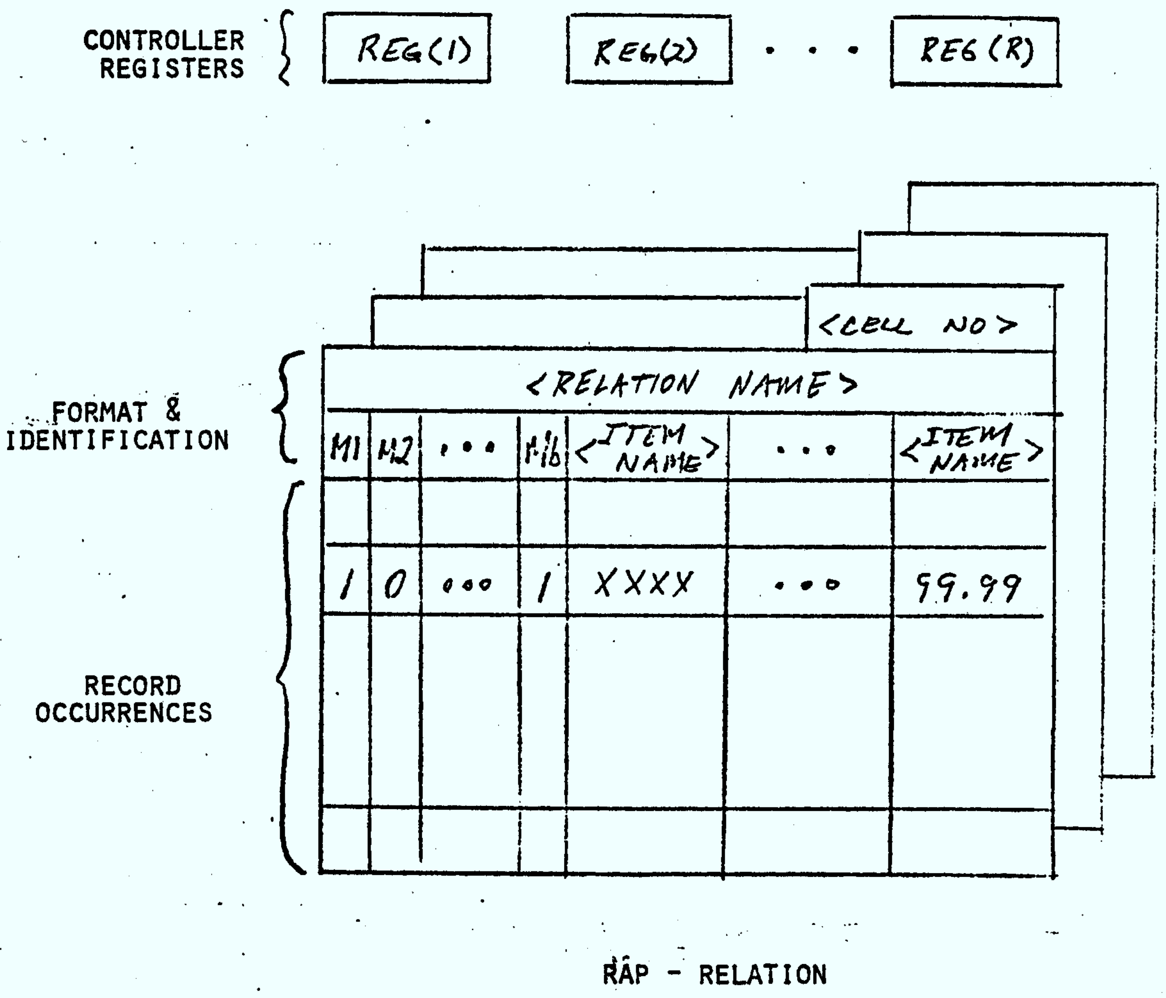


FIGURE 2 - RAP LOGICAL DATA STRUCTURES

The occurrences of a relation can occupy one or several cell memories, but each cell can only store occurrences from one relation. Therefore, a single RAP device can contain occurrences from one large relation or from N relations, one for each of N cells. The programmer of a query need not be aware of the cell location or number of cells occupied by the relations. However, there are occasions, such as during garbage collection or bulk loading, where the user needs to control the device at a cell level. To permit this, a user can refer to registers containing an integer address for each cell.

Several registers are also available in the controller. These can be used to store intermediate computations or retrieved data from relations and used as search values or tested in subsequent instructions to executed complex queries.

A RAP relation is an intermediate-level abstraction of large data bases. Although it has a flat tabular structure, it is not quite relational as defined by Codd. For example, duplicate records are permitted and their existence is not automatically detected. There are physical limitations on sizes and numbers of items. Also, the special hardware operations for mark bit manipulation is a form of hardwired "access method" that a user must control via program instructions to select desired data for further processing. What RAP does is to provide a data view or model that is high-level but flexible or general enough to easily support software implementations of set-oriented versions of common user views such as hierarchies, networks, and relations. An outline of how RAP can do this is presented later.

2.2 Instruction Format

The general format of most RAP instructions is:

<label> <opcode> <mark option> (<object> : qualification>) (<parameter>)

Exceptions will be noted as they arrive. The label is an optional symbolic instruction address, the opcode specifies the data manipulation operation. A mark option can take one of the following forms:

a) <null>, implies no marking is done.

b) MARK <bit specification>, Sets (to 1) the mark bit data items specified in the bit specification of the qualified tuples.

c) RESET <bit specification>, resets (to 0) the mark bit data items specified by the bit specification of the qualified tuples.

The individual mark bits will be donoted M1, M2, ..., Mb where b is the hardware parameter limiting the number of mark bits. A bit specification is simply a list of mark bit names. An object has one of the following formats and is used primarily to specify which cells are to be manipulated by the instruction:

a) n(D1,...,Ds) where Rn is a relation name and (D1, D2,...Ds) is a list of data item names associated with relation Rn. The data item list is optional or not relevant in many instructions. The index s is a hardware limit on the number of domain names that can be included for certain instructions.

b) List of cell address, CELL(i), where i the integer address of the i-th cell.

A qualification in the RAP instruction format can take one of the following forms:

a) <null>,implying every tuple of the relation qualifies.

b) Q1& Q2& Q3 ... &Qp, denoting the conjunction of simple conditions Qi.

c) Q1 | Q2 | Q3 ... | Qp, denoting the disjunction of simple conditions Qi.

A simple condition Qi can be any one of the following:

a) <Di> <comparator> <operand>

where i) Di is a data item name

ii) comparator is one of: $=, \neq, <, \leq, >, \geq$

iii) operand is one of REG (i), <integer>, '<literal>', where REG(i) refers to the contents of the i-th controller register.

b) MKED (Mi) denoting the mark bit test Mi = 1.

c) UNMKED (Mi) denoting the mark bit test Mi = 0.

d) CELL (i) indicating that the cell address is tested as part of the qualification.

A qualification has certain restrictions which are not apparent from the description of its syntax. A qualification can have at most k simple conditions of type (a) (i.e., data item comparisons) and 6 simple conditions of types (b) and (c). Only one simple condition of type d may be included in any qualification.

The format of parameter varies greatly and will be explained along with each instruction that requires additional information not supplied above.

2.3 Description of RAP Instructions

2.3.1 Selection

Select:

<label> Select <mark option> (Rn : <qualification>)

This instruction selects qualified tuples from the relation Rn and sets or resets the mark bits of these tuples according to the mark option given. For example, the instruction:

Select Mark (M1M2) (R1 : D1 = 'a')

will set mark bits M1 and M2 of tuples in R1 which have D1 = 'a'. Whereas the instruction:

Select Reset (M1M2) (R1 : D1 = 'a')

will reset the mark bits M1 and M2. A null operation occurs when the mark option is omitted.

Cross Select:

<label> Cross-Select <mark option on R1> (<R1>.<D1> <comparator> <R2>.<D2>)
(<R2> <mark option on R2> : <qualification>)

This instruction involves operation between two relations called source (R2) and target (R1). It works like a repetitive select instruction on the target relation with the exception that qualification for each selection is obtained from the source relation data item values. That is, in order to select a target relation (R1) tuple, the items D1 and D2 respectively of target and source relation must have comparable values (i.e., values from the same domain) that satisfy the comparison between them.

The source tuples participating in the comparison are those which satisfy the second qualification.

For example:

Cross_Select Mark (M2) (R1 : D1 < R2.D2)

(R2 Reset (M1) : D4 = 8)

Source tuples participating in the comparison are those which satisfy the qualification D4 = 8. Notice the M1 mark bit of the participating tuples of the source relation will be reset due to the mark option on R2. In order for tuples of R1 to be M2 marked, it must have values of the D1 item less than the value of one D2 item of the participating source tuples. To further illustrate this, suppose R1 and R2 have the following values:

R1 (target relation)

Tuple #	D1	-	- -
1	15		
2	6		
3	9		
4	30		
5	1		

R2 (source relation)

Tuple #	D2	D4	- -
1	10	7	
2	4	8	
3	11	8	

The participating source tuples are tuple 2 & 3 from R2 since they satisfy the condition D4 = 8. The tuples of the target relation that will be M2 marked are tuples 2, 3 and 5 because they meet the condition on R1 that D1 < R2.D2. A null operation will result if both mark options of R1 and R2 are missing.

2.3.2 Retrieval

Read All:

<label> Read-All <mark option> (Rn (D1, ..., Ds) : <qualification>) (<work area>)

This instruction transfers data from all tuples of Rn satisfying the qualifications to the supporting processor's storage address as specified by work area. This could be a seuqence of primary memory addresses or a file designation. If the object data item list is present, only those item values are read out, otherwise, the entire eligible tuple is transfered. If the mark option is present, the mark bit items of the eligible tuples will be set or reset according to the given mark option. For example:

Read-All Reset (M1M2) (R1 (D1,D2) :mked (M1) & mked (m2) & D3 = 10) (file.0)

Eligible tuples in this case are those which are M1M2 and marked and have D3 = 10. Notice only D1 and D2 of eligible tuples will be read into file .0. M1 and M2 mark bits of the eligible tuples will be reset.

<u>Read</u>:

<label> Read (n) <mark option> (Rn (D1,...Ds):<qualification>) (<work area>)

This instruction is very similar to the Read-All instruction, except that only data items from the "first" n or less qualified tuples are transfered to the supporting processor's storage location. The mark option will only be exercised on the tuples that are transfered.

<u>Save</u>:

<label> Save (n) <mark option> (Rn(D1,...,Ds):<qualification>) (<register list>)

Save (n) transfers data items from qualified tuples of a relation to registers of the RAP controller. Only items from the "first" n or less eligible tuples are transfered. If the mark option is present, the mark bits of the tuples will be set or reset according to the mark option. If the data element list is not present, the entire tuple will be transfered, otherwise, only those items in the list are read into the registers. Values will be stored left justified and padded on the right with blanks. Data elements with arithmetic domains will be assumed to be a fixed word length in twos-complement format.

Register list can take on any combination of the following 2 forms:

a) Reg (i), Reg (j),..., Reg (k)

b) Reg (i) - Reg (j), i < j

where Reg (i) - Reg (j) means Reg (i), Reg (i + 1),..., Reg (j). The transfer is done in the order given, that is, the first item in the object list is read into the first register designated in the register list, second item into the second register, etc. The n items are read from each tuple, the first item of the second eligible tuple will be read into the n+1 register in the register list.

Read Reg:

<label> Read_Reg (<Reg_list>) (<work area>)

This instruction transfers contents of the specified RAP registers to the supporting processor. Register list has the same format as the register list in the save instruction.

2.3.3 Statistical Computations

Sum, Count, Max, Min:

<label> <sopr> <mark option> (Rn(Dn):<qualification>) (Reg (i))

where sopr is one of the statistical function operators sum, count, max or min. The opcode count counts eligible tuples in the relation Rn and places the result in the register specified. (Dn) is omitted for this statistical function. The other instructions compute the specified function over the numeric domain of item Dn from qualified tuples. Mark bits of the qualified tuples can be set or reset by the presence of the mark option.

2.3.4 Update

<label> <opr> <mark option> (Rn(Dn):<qualification>) (<opd>)

where opr is one of the operators add, sub or replace and opd is either a constant, a data item name, or a RAP register. Item Dn in every eligible tuple is operated on by opr. Like previous instructions, the mark bits of the eligible tuples can be set or reset by the presence of the mark option.

2.3.5 Insertion and Delection

Delete:

<label> Delete (Rn : <qualification>)

Tuples of relation Rn qualifying for deletion has their delete flag bit set causing the tuple to be ignored in subsequent operations.

Colgrbg:

<label> Colgrbg (<relation list> and/or <cell list>)

This instruction initiates the physical deletion of all delete-flagged tuples of thelisted relations and/or listed cells. The data is packed towards the beginning leaving garbage accumulated towards the end of the cell memory. The cell list has the same format as a register list. The relation list has the following format:

(R1, R2, ..., Rn)

Space Count:

<label> Space_Count (Rn : <cell list>) (Reg (i))

This instruction will examine the cells of relation Rn and returns a value indicating the number of available spaces in these cells. This value is stored into the given register. Available spaces include both empty tuples and the delete-flagged tuples. If the optional cell list is present, only those cells in the cell list will be examined. All cells in the cell list must belong to relation Rn. This instruction is usually used to test for space before an Insert instruction is used.

Insert:

<label> Insert (n) (Rn : <cell list>) (<work area>)

Work area is the front-end processor's program storage location containing the n tuples to be inserted. If the optional cell list is given, the n tuples will be inserted in those cells only. There is an arbitrary hardware upper limit on the number of characters that can be inserted in one INSERT instruction which places a limit on n.

2.3.6 Data Definition

Destroy:

<label> Destroy (Rn : <cell list>)

This instruction deletes the tuples, format, and names of the specified cells of a relation. If a cell list is not present, the relation is removed from all the cells it occupies. A special null relation name is reserved for all blank cells.

<u>Create</u>:

<label> Create (Rn : <cell list>) (<format>)

One execution of this instruction formats each cell in the cell list for relation Rn. Empty tuples are delete flagged on the created cells. Format contains parametric data about the length of the data items stored in a relation.

2.3.7 Register Manipulation

Only registers containing valid integer values will result in meaningful numeric computations. All register arithmetic will assume word length operands on the left-most bits of controller registers.

<u>Insert Reg</u>:

<label> Insert_Reg (<register list>) (<constant list>)

This instruction will insert the constants into the specified registers. If only one constant is present, this constant will be inserted in all registers of the register list. Otherwise the number of constants must match the number of registers.

<u>Dec Reg, Inc Reg</u>:

<label> Dec_Reg (Reg (i))

<label> Inc_Reg (Reg (i))

The instruction Dec_Reg performs the following operation:

Reg (i) = Reg (i) - 1

The instruction Inc_Reg does the following:

Reg (i) = Reg (i) + 1

<u>Register Arithmetic</u>:

<label> <ropr> (Reg (i)) (<ropd>)

where ropr is one of the operators; Radd, Rsub, Rmul Rdiv and ropd can either be an integer or another register.

2.3.8 Decision and Transfer

BC:

<label> BC <label> , <boolean expression of conditions>

where BC is the abbreviation for "branch on condition". Condition can be one of the following :

a) null -- this implies the instruction is treated as an unconditional branch

b) Reg (i) <comparator> Reg (i)

c) Reg (i) <comparator> <constant>

d) Test (Rn : <mark qualification>)

If the boolean condition is true, branching will take place, otherwise control is given to the next instruction.

Condition type (d) tests each individual mark bit of specified by the mark qualification separately and if the test is met for at least one tuple (not necesary) the same one) of relation Rn then the test is true, otherwise, the test is false. Mark qualification can be either disjunctive or conjunctive. For example:

Test (RI : MKED (M1) & MKED (M2))

is true if the M1 mark bit is set to one for at least one tuple and the M2 mark bit is set to one for at least one tuple (not necessary the same one), otherwise the test is false.

EOQ:

<label> EOQ

This indicates the end of a RAP program.

2.4 Summary of Performance

Execution times depend on the speed of the cell processor and the capacity of a cell memory. These could vary greatly depending on choice of technology and architecture of the processor and memory. Therefore, we give a summary in terms of the number of searches or scans of the memory are required to execute on instruction.

TYPE OF OPERATION	INSTRUCTION	EXECUTION TIME (IN # OF SEARCHES)
Selection	Select	1
	Cross_Select	1 + (# of source tuples/k)
Retrieval	Read-all	1 + transfer time
	Read (n)	2 + transfer time
	Read_Reg	0
	Save (n)	2
Statistical Functions	Sum	1
	Count	1
	Max	1
	Min	1
Update	Add	1
	Subs	1
	Replace	1
Insertion & Deletion	Delete	1
	Colgrbg	1
	Space_Count	1
	Insert (n)	1
Data Definition	Create	1
	Destroy	0
Register Manipulation	Insert_Reg	0
	Inc_Reg	0
	Dec_Reg	0

TYPE OF OPERATION	INSTRUCTION	EXECUTION TIME (IN # OF SEARCHES)
	Radd	0
	Rsub	0
	Rmul	0
	Rdiv	0
Decision & Transfer	BC	0
	EOQ	0

An important feature of the RAP instruction set is that it is relationally complete. This means that any query expressable by the relational calculus can be implemented entirely within the RAP processor. This eliminates the need to transfer extensive amounts of data derived from the intermediate results of query processing between RAP and the supporting computer.

It is important to note that each high-level instruction operates on at most two entire relations during its execution. The hardware is naturally locked during an instruction execution. Thus all software schemes concerned with mutual exclusion of update operations can simply implement synchronization mechanisms at the relation level. This eliminates much of the operating system overhead incurred by conventional implementations.

Studies have been conducted to compare the hypothetical performances of the original RAP architecture relative to a conventional computer system for implementing a relational data base (8). Both appoaches were modeled analytically. The models considered resident data bases for the original RAP architecture and fast access paths in the form of inverted lists for the conventional system. The results show that significant gains in query execution speed can be achieved by the RAP architecture over the conventional system. Furthermore, the newer architecture improves this gain substantially. The model studied queries of the form: retrievals and updates on rows of relations selected with respect to simple and complex boobon qualifications, retrievals that include statistical criteria in the selection qualification, and retrievals involving the implicit join of two or more relations. This study indicates that, under many circumstances, on-line retrievals and updates of large data bases may only be possible with the use of RAP-like Systems.

3. IMPLEMENTATION

3.1 History

The RAP project was started in 1975 in the computer systems Research Group at the University of Toronto in order to implement a data base system based on the on the relational model in hardware. It started with the direct implementation of the logic specified in Ozkarahan's thesis (4) and in 1976 resulted in a system (hereafter called RAP.1) consisting of two cells. The RAP.1 system consisted of a hardwired controller and where each cell had its own memory <u>track</u> where a portion of a relation is stored. The format and timing of a track is modeled on disk technology.

In RAP.1, all components of the system were required to be synchronized by one single clock, all tracks had to be of equal length, and inter-cell communication required data flow between all cells. Every operation concerning data on the track took one or many "revolutions"; a revolution is the time it takes to serially scan a track.

During the project two important decisions were made to change the architecture of RAP.1 which resulted in the design and implementation of RAP.2. First, the controller was to be implemented by a mini/micro computer. Second, the data track was designed around the capabilities of emerging block addressible technology instead of disk. A hardwired implementation of the controller was found to be inflexable and speed was not an issue. The Development of a disk system that meets the requirements of a RAP system was difficult because of synchronization and error correction problems. Furthermore, there is a widespread belief CCD, bubbles and/or electrom beam technologies will eventually cause disks to be phased out.

The use of a general purpose computer as the controller resulted in dramatic changes in the RAP architecture. Due to the flexibility of the computer, a major redistribution of the work-load became feasible. In RAP.2, the cells are greately simplified and asked to do only those tasks related directly to their tracks while the controller takes care of the rest. Because the controller is inherently slow and cannot cope with the speed of the cell, it became important to decouple cell synchronization from the controller. The new work-load distribution also frees every cell from the task of sending data directly to other cells. This can be done through the controller. Each cell can operate independently of other cells and the controller. As a by-product each cell can have its own track length. RAP.2 looks like a conventional computer system with the controller coordinating the tasks of many cells which are treated as independent peripheral devices attached to bus lines.

In summer of 1977, a RAP.2 prototype of 2 cells and query language software were demonstrated. Each cell contained a million bit CCD track built from Intel's 2416. The controller was a PDP11/10. The RAP.2 was interfaced to a PDP11/45 as a DMA device. To make the transistion from RAP.1 to RAP.2 as fast as possible, it was decided that only essential changes were allowed. Consequently, RAP.2 is far from perfect and its performance can be greatly improved. In this paper we refer to enhancements not yet implemented as features of a future RAP. 3 system.

3.2 Physical Data Organization

In review, data is organized into files called RAP relations. A relation is a collection of records called tuples. Each tuple is a string of many concatenated fields called data items in some fixed order. The number of fields per tuple of a given relation is a constant. Every relation and each of its fields have a name stored in a compactly coded form. In RAP, the length of each item must be constant. In other words, all tuples of a relation share a common FORTRAN-like format. In the RAP.1 and RAP.2 prototypes, each tuple can have up to 255 domains and the length of a domain can be either one, two or four bytes of encoded data. In RAP.3, this length can be anywhere from one to n bytes (where n should be 32 bytes or greater). Each cell stores only one relation. If a relation has too many tuples. thev can be allocated in manv cells.

In RAP.1, a cell stores the relation name as well as the cell address at the track head followed by tuples separated by gaps (Fig. 3). Each gap must be at least the size of the longest domain and furthermore, must be such that the block size (i.e., tuple length + gap length) is either 256, 512, 768 or 1024 bits. A tuple is preceded by 7 mark-bits and ended by a 2-bit length code (Fig. 4).

In RAP.2, the cell address is defined by an 8-contact switch set by an operator. The relation name is stored in a 16-bit register and is defined by the programmer. Both the cell address and the relation name can be read out. The CCD memories of each cell behave like a very long drum with many small tracks of 256 bits each. Due to historical reason, in the remaining part of this paper, by "track" we mean the entire CCD drum and each 256-bit circumference will be called a minor loop. The format for each tuple remains unchanged except that the two-bit space between 2 consecutive domains are left blank since all the length codes are stored in a register called the LENGTH CODE RAM. As for gaps, the only requirement is that each tuple must fit in an arbitrary integral number of minor loops (Fig. 5). RAP.2 simulates a disk read head by the use of a counter which points to the "current" location. The write head can be calculated from the read head bv usinq an adder. For most instructions, the write head is one block behind the read head, in some instructions the two heads are identical. Because of the randomly accessible nature of minor loops, access time is small (the worst case is 256 bit-times). When a cell is not doing anything, it is "on" the first minor loop. In operation, each instruction requires the heads to scan just enough space to complete the job. After an instruction is completed, the heads immediately return to the first minor loop. Due to this property, it is more appropriate to use the term "pass" or "search" instead of "revolution" to indicate the time required to do an instruction. In the worst case, a pass is one revolution. Typically, a pass is porportional to the length of the track portion containing data. Sometimes, as in data retrieval or insertion, a pass

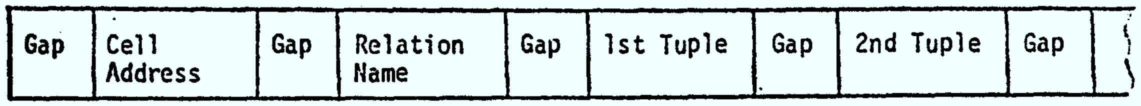


Fig 3. RAP.I's data track.

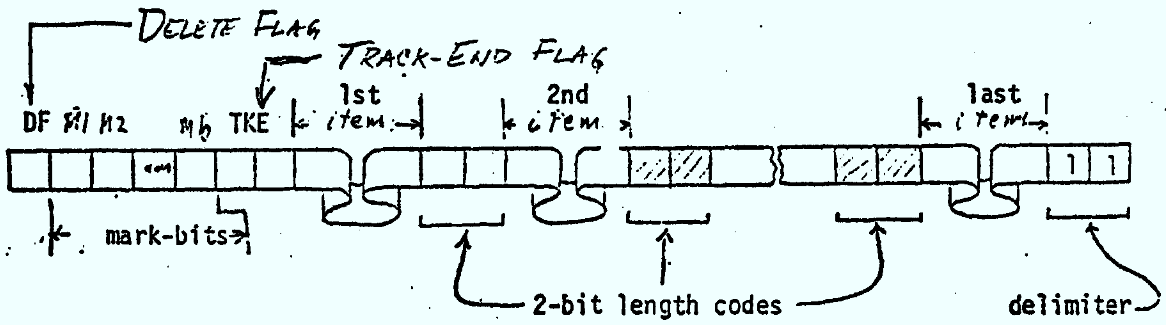


Fig 4. RAP.1 tuple.
RAP.2 still keeps the same tuple's space distribution but the 2-bit spaces after domains are left blank.

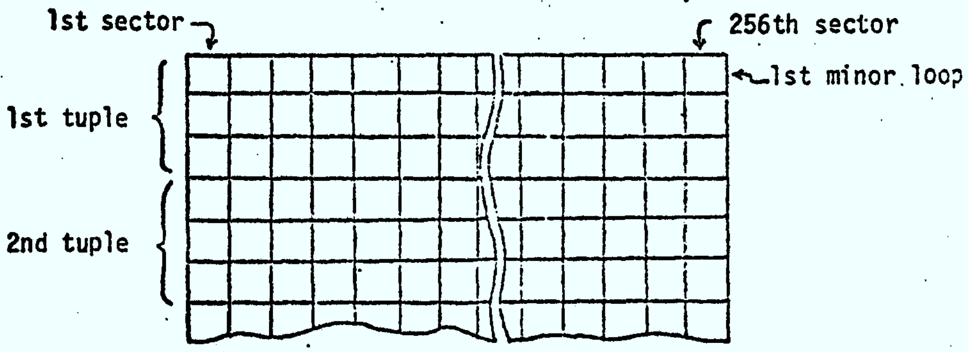


Fig 5. RAP.2 "dissected" drum.
Drawing shows a file with 3x256-bit tuples.

ends immediately when enough tuples have been retrieved or inserted.

In RAP.3, storage efficiency is maximized. There is no inter-item space and no gap. A more sophisticated delay mechanism is employed to write data in proper spaces. There is a "return from halt" option (analogous to the "return from subroutine" instruction of some microprocessors) that allows the cell to resume scanning at some previous spot. This option greatly improves execution time where a very large volume of data is inserted or retrieved.

3.3 Global Architecture and Communications

The RAP.2 system is organized as shown in Fig. 6. There are eight control lines and data is exchanged between the cells and controller via a bilateral 16-line bus. There is no direct data link between cells. A DMA link is established between the data bus of the controller and front-end computer.

There is a priority line that runs through all cells to allow fast pulling of individual cells. This is used to control cell access to retrieved intermediate computation, retrieval qualified records or perform bulk loading. The line is not essential but in a large RAP system, the it reduces access time and storage space in the controller. As a variation it is also possible to "fragment" this line; the cells can be grouped where each group has its own priority line.

The reason why the direct data communications between cells lines was dropped in RAP.2 is multifold. First, expensive drivers were required because each cell was able to drive all others. Second, there was the classical transmission line problem requiring the entire RAP.1 system to be crammed into a physically small space. Third, we wanted to desynchronize the system. Last, reliability was questionable when data is sent automatically from any cell directly to all others. If one cell is malfunctioning, the whole system would suffer and diagnosis would have been an extremely difficult task. Furthermore, the amount of information to be exchanged was usually too small to justify the cost of any direct communication link.

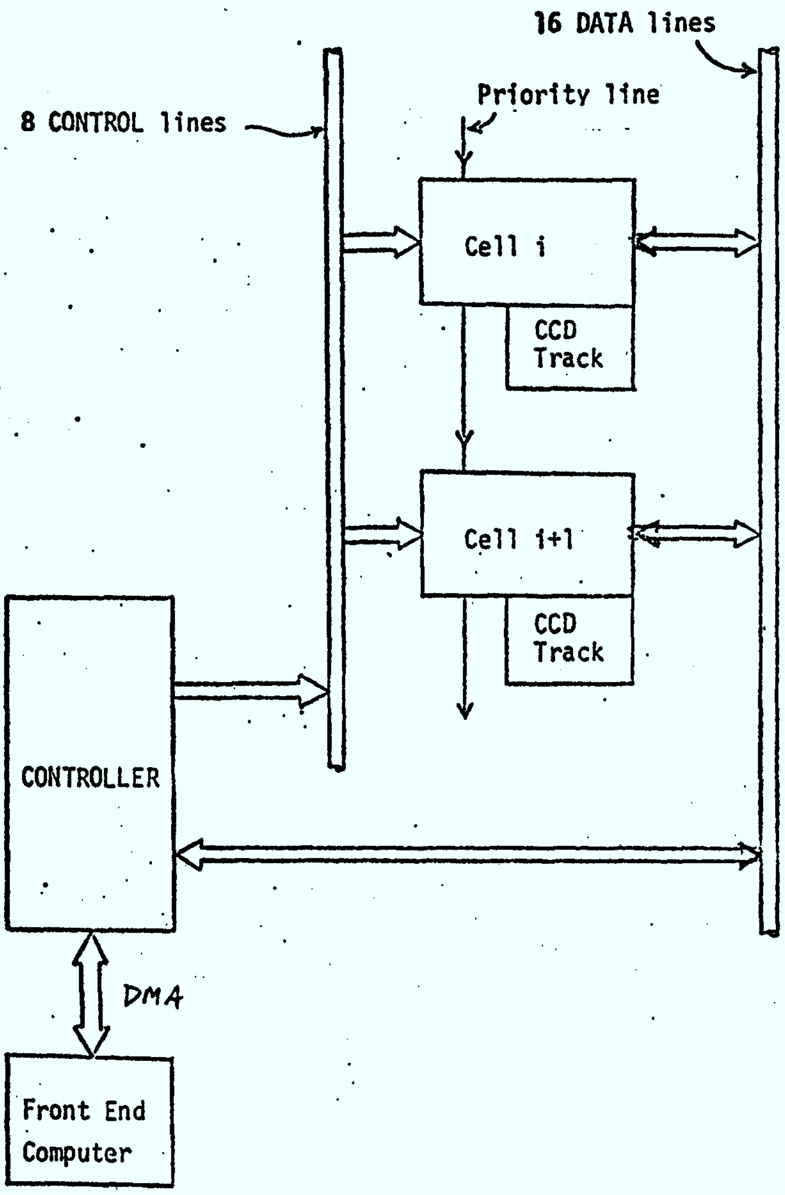


Fig 6. RAP II System Structure.

Seven of the control lines allow a maximum of 128 micro-code commands called "keys". The eigth control line is named "key enable" and is similar to the "valid data/address" line of microprocessors. In the absence of key enable, the cells ignore everything on the bus lines. Each combination of the seven control lines is interpreted as a command. In practice, these lines are connected to the least significant part of the address bus of a PDP 11/10 controller and the key enable line is decoded from the most significant part and from some other lines. Therefore, each reference to one of 128 ficticous memory locations are reserved for RAP and is interpreted as a key. Some keys are accompanied by an operand which must appear on the data bus, some expect data from cells to be put on the data bus and others are not associated with any data at all.

Commands are broadcasted indiscriminately to all cells of the system. Establishing a scheme to restrict selectively the commands to only one or a few cells is handled by the creation of three state-variables called "open", "blocked" and "rejected". A cell must be open but neither blocked nor rejected to be sensitive to all commands; it ignores most commands otherwise. There are three different ways to open a cell. In the simplest case, the controller can open all cells simultaneously by referring to special keys. The controller can also open any particular cell by its integer designation by storing that value in one key location. Finally, cells can be opened by referencing the name of the relation stored in the cell.

Under some circumstances, it is necessary to restrict the communication to only one cell. For example, in Insert, it is not desirable to have a same tuple to be written in several cells of a same relation. A simple way to achieve this is to open by cell address. This requires, however, that the controller either has to keep track of an address table or to poll all the cells one after another and is therefore suitable only in small systems. The states "blocked" and "rejected" together with the priority line and the "get next cell" command are intended for this purpose. Consider the following analogy. A number of persons forming a line to buy a ticket in a theatre. Those who have bought one are "rejected"; those who still have to wait are "blocked". The one who is buying is neither blocked nor rejected. Each time the line moves corresponds to "get next". Only open cells are sensitive to a get next cell key. This command rejects the current non-blocked cell and unblocks the first blocked cell. For example, to serve all cells of a relation Rn sequentially, the controller must first open all Rn cells and then block them which is achieved by reffering to another location. It then sequences through a program loop starting with a get next cell and followed by the service routine.

There must be some way allowing the controller to know if there is any cell "listen" to its commands. This is possible by assigning the value "1" to the most significant bit of a cell status register which can be read either selectively or collectively. The controller simply has to read the status to know if its audience is non-existent.

For a cell to respond to a command, the cell must be in a proper state: it must be open, neither blocked nor rejected, and furthermore, not running. The last condition is a measure of protection against any erroneous attempt to change the parameters of a query or the nature of the instruction being executed. Also, the controller has to store relevant data into appropriate reserved memory locations. Many single bit items are grouped in bytes to save time and space.

For I/O each cell has a (1K-word) RAM called the I/O buffer and a pointer which is resettable by the controller. As far as the controller is concerned, for loading, the I/O buffer looks like a single reserved memory location. Every time a word is stored in this location, it is sent to the I/O buffer where pointer is incremented automatically. In this implementation, to insert a set of tuples, all the controller has to do is repeatedly store two bytes at a time in the reserved location.

It indicates how many tuples to be inserted by writing this number in another reserved location. After the cell is initiated to run, it will look for enough vacant slots on its track and pull data from its I/O buffer to fill them.

During data retrieval, the opposite is done. The cell looks for desired data on its track and puts them in its I/O buffer. Since the buffer size is limited, the controller must also indicate how many tuples to be retrieved. As far as the controller is concerned, for reading, the I/O buffer also looks like a reserved memory location. The buffer pointer is automatically incremented every time this location is read out.

Besides track data, many other kinds of information of a cell are also available for retrieval by the controller: processing status, cell address, relation name, buffer pointer, result register for computations and the S-counter which contains the number of satisfied tuples in the most recent pass. Access to the buffer pointer allows the establishing of a future DMA link with any cell attached to the data bus for mass transfer of inserted/retrieved data. This could permit DMA transfers directly between cells and the front-end computer.

3.4 Instruction Set Execution

An important feature of RAP.2 is that many micro-instructions can be overlapped or carried out simultaneously. After the controller has initiated some cells to do one instruction, it is free to prepare other cells for a new instruction. Since a pass is typically much longer than preparation time, there is virtually no limitation to the degree of concurrency. Software is being developed to exploit this property.

Each macro-instruction is divided into many smaller parts called "tasks". There are three distinct classes of tasks:

(a) The principle class consists of tasks involving the scanning of a track (e.g., writing on a track, finding a blank space on a track, etc.) These tasks are performed strictly by the cells after being initiated to run by the controller. Every cell determines when to stop the execution of a track (e.g. when the physical end of the track is reached). Each task is completed in one single pass which is, under the worst condition, one revolution time.

(b) The second class consists of tasks done by the controller to the cells (e.g., opening cells, sending operands, etc.). As mentioned before, each task of this kind is carried out by referring to one of the reserved memory locations. Typically, each task of this type takes 1 μsec to be executed.

(c) The last class does not involve the cells (e.g. register manipulation, DMA communication, etc.). These tasks are accomplished within the front-end computer and/or the controller.

3.5 Cell Structure

The structure of a cell can be divided into eight units as shown in Fig. 7. In the following, we will describe briefly the functions of each.

Cell Interface:

This unit is the part of a cell that "listens" to the outside world. It contains bus receivers and a large decoder that decodes the contents of the control bus. This decoder is prohibited by some states of the cell to make restricted communication feasible. Some of the outputs of the decoder are for steering data from the data bus to appropriate registers. Others are for reading from various places of the cell or for changing states.

The cell address is part of this unit and is defined by an 8-contact switch. The switch setting is constantly compared with the data bus by an address comparator. If matching occurs while the controller is referring to a reserved location the state "lock open" will be high indicating that the cell is open. Once a cell is open, it remains so until specifically told to close.

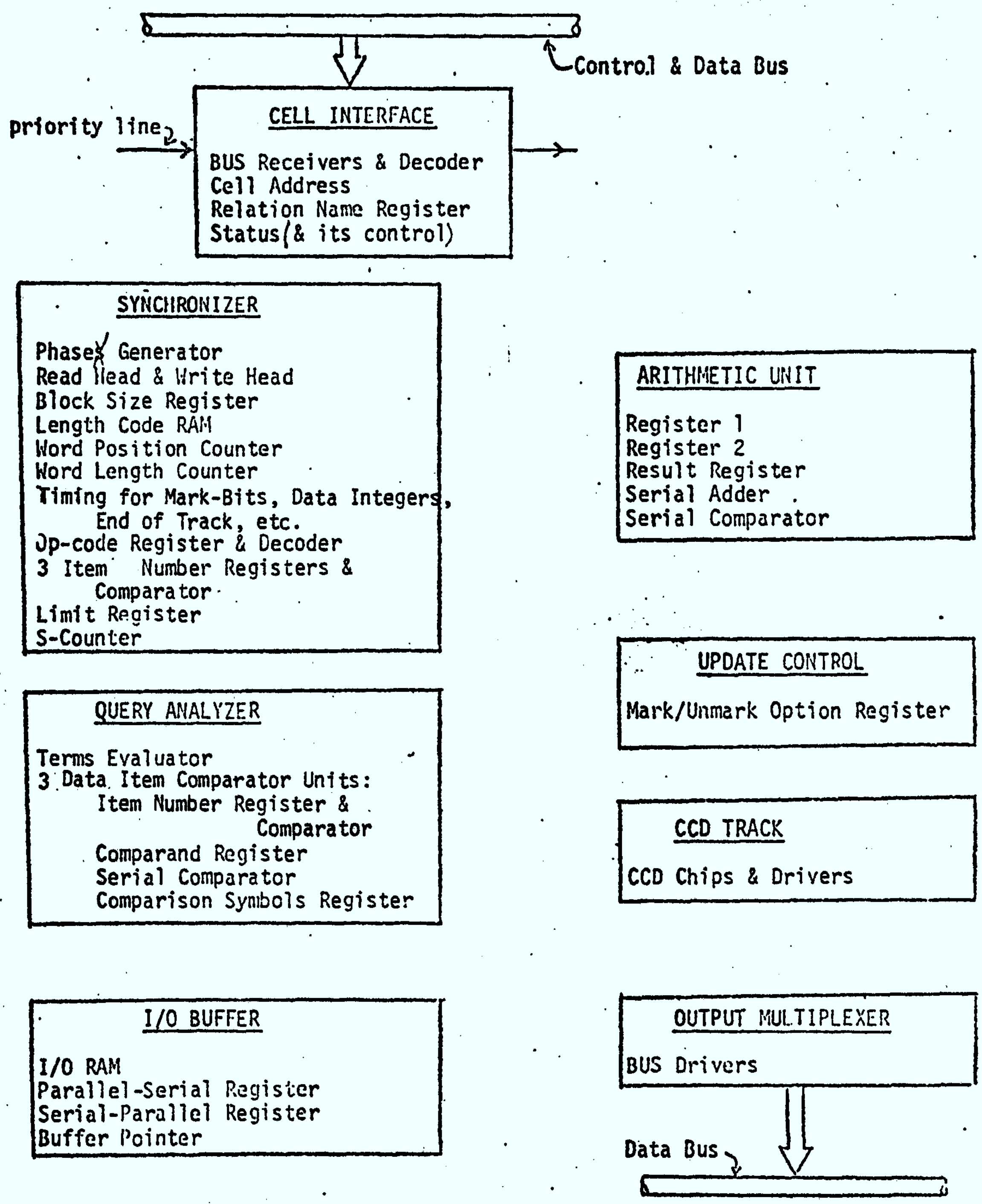


Fig 7, RAP.2 Cell's Block Structure

There is a 16-bit relation name register which behaves much the same way as the cell address. The main difference is that it is settable by the user via the controller. Another difference is that if the cell is "destroyed" the relation name comparator is negated, thus opening by relation name becomes impossible. When the cell is given a new name in a future CREATE instruction allowing again opening by relation name.

The logic for "get next cell" is also part of this unit. As mentioned before, every reference to a specific location will affect the states "blocked" and "rejected" of an open cell.

Finally there are also status states indicating whether, in the last pass, there is a mark on bits DF, M1, M2, etc. and a state indicating if there is a satisfied tuple. The most significant bit of the status is always a "1" and is used to indicate the presence of a cell.

<u>Synchronizer</u>:

This is the largest logic unit of a cell. It provides all timing signals and shift clocks to the rest of the cell except the output multiplexer which does not need any. For simplicity and ease of testing, all basic clock signals are periodic. This implies that there is always a read phase and a write phase for the CCD memories. Consequently, the bit rate of RAP 2 is slightly below 1 MHz. (In RAP 3, read and write phases would be allowed only when necessary). The 1 MHz bit rate is a limit of the CCDs not the cell logic which has been rated at 10 MHz.

There is an 8-bit sector counter that keeps track of the current position on a minor loop. Another (extendable) counter points to the current minor loop. These two counters simulate the read head. The write head is calculated from the other by subtracting the contents of the block size register. A time multiplexing logic allows switching between the two heads. The block size register is loaded by the controller with the size (number of minor loops) of a tuple during a Create process. For self-testing purpose, some logic has also been incorporated in RAP.2 prototype cells to halt cells to implement the "single step" mode or to slow them down. This is possible by an appropriate manipulation of the read head.

Also, in a Create process, the format of the tuples must be loaded in a 256x2 RAM called the length code Ram (LCR). This allows a maximum of 256 data items per tuple and at most 3 different size of domains. In RAP.3, the width of the LCR is expanded to allow more choices. In operation, the LCR is sampled just before any domain is expected. The contents are passed to the word length counter which decides when to expect the end of the current domain and hence the next domain. These contents are also decoded and sent to the feedback control of various variable-length shift registers to allow their rotation.

Every time the LCR is loaded or sampled, its pointer is incremented accordingly. The pointer is the contents of the word position counter which keeps track of which item is under scan. There are three 8-bit registers storing the data item number of up to three "specified" data items to be read out or that are operands in an arithmetic operation. A comparator is used with the word position counter to test sequentially the three registers to determine when a specified item is being scanned.

An important duty of the synchronizer is to stop the cell at the end of an instruction. As mentioned before, each instruction ends in a distinct way. There is a 4-bit op-code register associated with a reserved controller address storing the code of the current instruction plus the op-code decoder. The S-counter constantly gets incremented every time a satisfied tuple is found. Typically, a "satisfied tuple" means a tuple satisfying the qualification; but this is not always true. In particular, Insert and Space-Count look for vacant tuples and Create for spaces being large enough to define a tuple. Thus, after Space-Count and Create, the S-counter indicates the number of vacant spaces on the track;

after Insert it indicates the number of new tuples just inserted. Due to the finite size of the I/O buffer, only a limited number of tuples can be read/inserted per pass. This number must be supplied to the limit register. This register is constantly compared with the S-counter to determine when enough satisfied tuples have been processed so that the instruction can be terminated (for Insert and Read only). Other instructions except Space-Count terminate when the logical end of the track is reached. The logical end is indicated by a vacant tuple bearing a mark (TKE) on its 6th and 7th bits. If the whole track is used, no logical end exists and all instructions terminate when the physical end of the track is reached.

Finally, there are several single-bit registers that sample the value of the mark bits that are required for the query analyzer.

Query Analyzer:

This is the heart of a cell and is the only unit that remains essentially unchanged from RAP.1. It determines whether a tuple satisfies the qualification. A query list has one of two forms: either disjunctive or conjunctive. A tuple qualifies if either no term is false (conjunctive form) or at least one term is true (disjunctive form). RAP..3 in addition allows the comparison of two data items of a same tuple as well as other forms of qualification.

The QUERY ANALYZER has two parts: the terms evaluator and three identical data item comparator units. Each comparator unit has an 8-bit register to store the item number of the relevant item a tapped 32-bit shift register for the externally supplied constant, a 4-bit register which indicates the selection of the unit and the symbols (<, =, >) of the comparison and a serial comparator. In operation, a comparator is constantly used to compare the item number with the contents of the word position counter to signal the scanning of the selected item. When this happens, the constant is rotated and compared with data read from the track. By the end of the selected item, the serial comparator knows the truth value of the simple condition of the query list covering this data item.

The terms evaluator has a multiple single bit register to store the form of the query list as well as the criteria concerning mark bits. It also has a small logic to determine, from the rest of the query analyzer, if the current tuple qualifies.

RAP.3 would not use shift registers to store constants (comparand in the comparator units and operands in the arithmetic unit). They are stored in a RAM for two reasons. First, long and tapped shift resisters are expensive. Second, there would be no need for a mechanism to rotate the registers. The pointer for this RAM is the contents of the word length counter.

<u>I/O Buffer</u>:

This unit is of prime importance to decouple a cell from the controller for data retrieval and insertion. The controller empties or fills up, at its convenience, the I/O buffer of each cell which loads or unloads its buffer whenever it can. It is not known what is the optimized size for the buffer, but the current prototype cells have a 1K x 16 RAM. Perhaps a size four times larger would be more suitable.

For data insertion, new tuples must be preloaded in the buffer. After the cell is initiated to run, it will look for vacant spaces which are marked at the 1st bit. When one is found, the cell erases all mark bits. New marks will be generated if the mark option is used. The data is pulled from the buffer. A 16-bit parallel-to-serial register is used to send one bit at a time to the update control unit for writing on the track. The cell stops when enough tuples have been inserted. If there are not enough spaces, the cell stops at the physical end of its track.

For data retrieval, data from the read head is shifted in a 16-bit serial-to-parallel converter to be loaded into the buffer either once per every 16 bits of a item or once per every 8-bit item.. If the tuple is found not to qualify, the cell back tracks its buffer pointer so that, in the end, the buffer holds only the selected items of qualified tuples. The cell stops when either enough qualified tuples have been read or when the logical or physical end is met. In RAP.3, since the read phase is allowed only when necessary, the retrieval procedure is different. The cell only reads the selected items of a tuple after it has been found to qualify.

Arithmetic Unit:

This is the only unit that is not vital to the operation of the rest of the cell. It is necessary only for supporting arithmetic instructions (namely Add, Sub, Sum, Max, Min) and can be removed if they are not required. It contains three tapped shift registers to store operands and results. For lack of space, the prototypes were built to support 8-bit and 16-bit operands only. It is trivial to expand these registers for longer fields. As mentioned in query analyzer, RAP.3 supports much longer domains where a RAM would be more suitable than shift registers. A serial comparator similar to the one of the item comparator units helps to implement Max and Min. A serial adder is used for Add, Sub, and Sum.

In operation, if the instruction involves only one item, the arithmetic is immediately performed while that item is being scanned. If two items are involved, the first one is stored in one of the shift registers until the second one is scanned. In Add, Sub, and Replace, the result is immediately stored in the result register to be written in the proper space later (i.e., during the next tuple-time) if the current tuple qualifies. For Max, Min and Sum, the result is transferred to the result register only after the tuple has been found to quality (i.e., in next tuple-time). It is up to the controller to probe the result register after the cell stops. Multiply and Divide are not implemented directly in RAP.2 cells because of timing conflicts in the bit serial logic.

Update Control:

This is the smallest unit of a cell. It has a register to store information concerning the Mark and Reset option. It takes care of the marking and resetting of mark bits as well as the writing of new data supplied by the I/O buffer for Insert or by the arithmetic unit for Add, Sub and Replace. It also erases the track for Create or only selected tuples for Delite. It also erases the track for Create or only selected tuples for Delite.

Output Multiplexer:

This is logically the simplest unit. Appropriate registers are connected to various bus drivers which are enabled by signals from the cell interface decoded from the control bus.

For most registers, it is the duty of the controller to assure that only one cell at a time is in the readable state otherwise information on the data bus is meaningless. The only exception is the reading of the status which is meaningful in an "OR"form..

CCD Memories:

This unit is built in a very straightforward manner. Due to physical limitations, the 1-megabit drum occupies three identical boards. Each board contains all necessary drivers and 20 Intel 2416 CCD chips which are arranged in an X-Y matrix of 4 x 5. This arrangement is necessary to reduce the number of drivers. It is possible because the CS and CE inputs of the chips are internally "and-ed" together. Two different kinds of drivers are used: Intel's 5244 chips are used for shift inputs and Intel's 3245 chips are for addressing, these drivers are quite good but there is a small problem of mismatched speeds. The former type is about 70 ns slower and some compensation must therefore be made. Currently, all the CCD chips are driven at a same frequency. If the memory size is expanded much larger, it makes sense to use two different rates. Only one or two chips at a time are driven at the fastest rate and the rest at the minimum frequency to conserve power.

3.6 Some Statistics

Each cell (including the 1-megabit track) requires about 9 amperes at 5 volts, is spread over 13 boards and employs 412 IC packages (218 SSI + 117 MSI + 77 LSI). For each additional 1 megabit extension, 96 IC's are needed.

4. DBMS UTILIZATION OF RAP

In this section we address four issues relating to total data base management system architecture and various uses of RAP as a systems component. First we consider systems approaches for data bases larger than RAP capacity. Next we show how RAP can be programmed to support multiple views of data. Last we consider RAP's applicability to protection, security, and integrity.

4.1 System Organization

One might conceive of the day in which microprocessor logic and memory becomes so inexpensive that all secondary memories would have RAP-like processing capabilities. However, the first generations of commercial RAPs would have capacity limitations relative to the total data base storage requirements. In this case, all existing data base installations have significant capital already committed to conventional secondary memories.

A complete cost effective system would therefore consist of a triad of component types: a front-end general purpose computer to interface with users and provide operating system and language processing functions, a RAP device used as a peripheral or back-end processor/memory, and one or more conventional secondary memories. The entire triad can also be considered as a back-end data base computer for further levels of front-end systems (11). We will briefly outline three approaches to the DBMS organization that exploit this organization at varying levels of complexity.

It is important to note that some of the problems alluded to in the section on limitations of conventional approaches are reintroduced with this approach. However, because the complex situations are handled by RAP, we would expect that an overall improvement in performance and a reduction in administration complexity would still be achieved.

4.1.1 RAP as an Access Path Processor

As stated before access paths are indexing methods used in conjunction with conventional DBMS implementation to speed the searching of large data bases. Two major types of access paths will be considered: inverted lists and links. An inverted list is a data structure which extracts the values that occur in the data base and associates with each unique value the physical or relative addresses of the records that contain that value. The records which satisfy a boolean selection criterion of simple item conditions of the form "<item> = <value>" can be found by processing the inverted lists rather than searching the data base directly. The boolean AND operator between two conditions is satisfied by the intersection of the inverted lists addresses associated with the item <values>. The OR operator is satisfied by the merge or union of the address lists. The NOT operator can be implemented by the relative complement of the list's address. Arbitrary boolean expressions can be parsed into a trees and processed two or more lists at a time producing intermediate lists until the final list contains the addressed of the qualifying records.

It is possible to create an inverted list on several different items within a file at the same time. This is done by associating the addresses of the records in which a particular combination of values occur where each value is from a different item. This type of inverted list is called a multi-key inverted list. In this case a conjunctive boolean qualification has already been formed as part of the list structure. The records satisfying a general boolean qualification can be found by first placing the qualification into disjunction normal form and since there exists a list for each conjunction, the qualifying records can be computed by just merging the lists to satisfy the disjunction.

In dynamic update situations where the inverted lists are stored on conventional devices, the single values inverted lists are preferred because they are easier to maintain. In static non-update environments, the multi-key inversion gives better retrieval performance.

A link is a data structure which records instances of pairwise inter-record connections of records of different type. It is often used to record the existence of a particular relationship between the records and provides a fast access path for locating and accessing the linked record given the other record. Such structures are used to optimize join operations in relational data bases system or the implementation of sets in CODASYL Systems (10).

Processing Inversions:

Inversions can easily be stored in RAP relational format. For a single item inversion we could create a RAP relation for each item inverted. There will be two entries in the records -- one for the value and one for the address pointer. For multi-key inversions one would create a RAP relation containing an data item for each value and one for the address. In cases where the same value(s) occur at many addresses, we must duplicate the values for each address.

Two things affect the efficiency of RAP for processing inversions: the size of the record occurrences and the type of processing required. In the single inversion case, each record has only two items - a value and an address. Because a gap is sometimes required between each record, the utilization of space would not be very efficient (this is eliminated in RAP,3). Secondly, the processing of single valued inverted lists required intersections or unions of data items between relations on RAP. This requires cross marking operations which are among the slower multi-revolution execution instructions for large volumes of qualified RAP records.

The above processing is to be distinguished from the union and intersection of items in the same relation which is very fast on RAP. This fact can be exploited to solve both the space and processing problems by utilizing one multi-key inversion for each data base file. The storage of more values per RAP relation creates larger and fewer RAP records. Also the clustering of lists into one RAP relation representation of the multi-key inversion creates a structure efficient for RAP processing. As stated before all conjunctive queries are preprocessed by construction of the list. Conjunctive queries involving only some of the inverted items are processed naturally by the RAP marking and qualification logic. Disjunctions are then simply processed by reading the address items of the marked records. However, it should be noticed that the list of address is not guaranteed to be in sorted order unless the storage or reading process is controlled.

Processing Links:

A simple RAP relation for implementing links requires a pair of items. Each RAP record records the addresses of the pair of linked data base records where the first address item always points to one data base file and the second points to the other. By knowing the data base address of a record one can mark the RAP linkage records using the address as a condition on one of the items. All the addresses of data base records linked to first can be readily read.

In many cases the link would have been established from values already existing in the data base. These values could be added to the RAP linkage structure as extra items. The values can then be used to aid in updating the link on RAP when the data base is modified.

4.1.2 Data Base Partitioning

The major problem with the indexing approach is that the index data stored on RAP are duplicates of data on disk. Instead we can partition the data base files both horizontally, i.e. placing certain records on RAP and other on disk, and/or vertically placing clusters of data items on one device or the other. Because no duplication takes place, the technique is called device partitioning. Extra data items such as record id's may be required to link corresponding partitions.

This particular approach attempts to exploit the notion that not all data in a data base, at a particular point in time, requires the same processing capabilities. Data can be categorized by the system according to its usage characteristics and placed on the conventional secondary memories or RAP depending on processing requirements that best fit the data.

The implementation of such a system should include mechanisms for both user controlled and automatic migration of data between the various devices as usage of the data changes. Research into algorithms that exploit data base device partitioning is under way at the University of Toronto.

A request would be processed by decomposing it into RAP and Disk subqueries and first executing the RAP subqueries. Access would then be made to disk only if the request can not be entirely serviced by RAP. In this case the response from the RAP subquery would be used to minimize the search over the disks portion.

4.1.3 Paging and Virtual Memory

This approach mimics the techniques of paging operating systems to provide a virtual associative address space for a RAP device. This requres all the data in the data base to be stored according to RAP memory format. The data is then divided into pages the size of one RAP cell. All data base queries are translated into RAP processing statements. Before execution, each query is directed through a software monitor executing on the front-end computer. The principal tasks of the monitor are to maintain a table that gives the location of the pages for each data base file, analyze which pages are required to execute a query, and then page the necessary data between the conventional secondary storage devices and the RAP processor. The query is then passed to the RAP processor for execution. It would be optimum to have a direct path between the secondary storage devices and RAP so that pages would not have to be transferred through the front-end.

As opposed to the indexed or partitioned approaches, all queries will be executed entirely within the RAP processor. This requires that all the data for a query be small enough to fit on the RAP device at any particular time.

A detailed design of the proposed monitor has been outlined in a previous study (9). Many of the arthitectual extensions proposed to the RAP device to allow the overlapping of paging with processing are not required by the RAP.2 architecture. A GPSS simulation of the entire system was performed and the results were analyzed (3,7). Statistics were collected on the average response time for on-line queries for a population of Poisson arrivals with a fixed mean and specific sized RAP device. Response was studied with respect to average exponential processing times, average amount of data stored in a relation, total data base size, and uniform and exponential locality of relation references. Locality was defined as the degree to which short sequences of queries reference some relations more than others. It was found that no significant losses in performance will occur in user environments which exhibit some relative combination of the following characteristics:

a) Relations that occupy a small number of cells.

b) Query populations which exhibit long processing times relative to their paging requirements so that overlapping of processing and paging can be effective.

c) Query populations which exhibit a "significant" amount of locality.

4.2 Supporting Multiple Logical Views

The concept of RAP relations and linkage between record occurrences can be used in the implementation of each of the three common views or models of data:

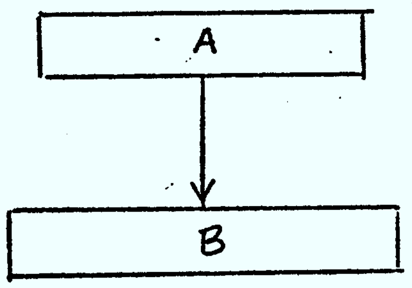
hierarchical, network, and relational. Relations viewed as tables of data have a natural correspondence to RAP relations. Many relational operations have a natural correspondence to RAP instructions (5).

Hierarchies and networks can be implemented by setting aside items in the record occurrences to store identification and structural data. We require that each record occurrence have one item whose value uniquely identifies the RAP relation and each particular occurrence within the RAP relation. This item will be called ID for identification.

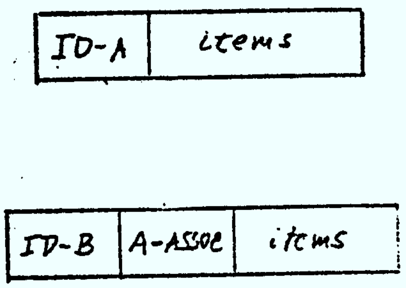
Implementing hierachies and networks requires the ability to implement functional associations between occurrences of record-types (17). A functional association can be defined as 1:N (i.e., one to many) linkage or mapping between record occurrences of two RAP relations. That is, if a 1:N linkage exists between RAP relations A to B then one record occurence of A can be associated or linked with zero or more unique records of B. Each record will have at most one A record for a particular association. One way to implement the association is to allocate an item called ASSOC in the RAP relation. This scheme is shown in Figure 8. For each record of B that is associated with one record A we store the record ID value of A in the ASSOC item of B. Finding the records of A with those associated with B and vice versa is simply a matter of using the cross marking selection instructions which interrelate two different RAP relations through the comparable ID and ASSOC values.

A second way to associate records of different type is to create a new linking RAP relation which contains two (or more) ID items -- one for each record-type. This is similar to storing link access paths on RAP except each occurrence of this RAP relation associates one record of A to one record of B by storing the associated ID's of the two records in one occurrence. This scheme has the advantage of implementing M:N (i.e. , many to many) associations between two types of RAP relations.

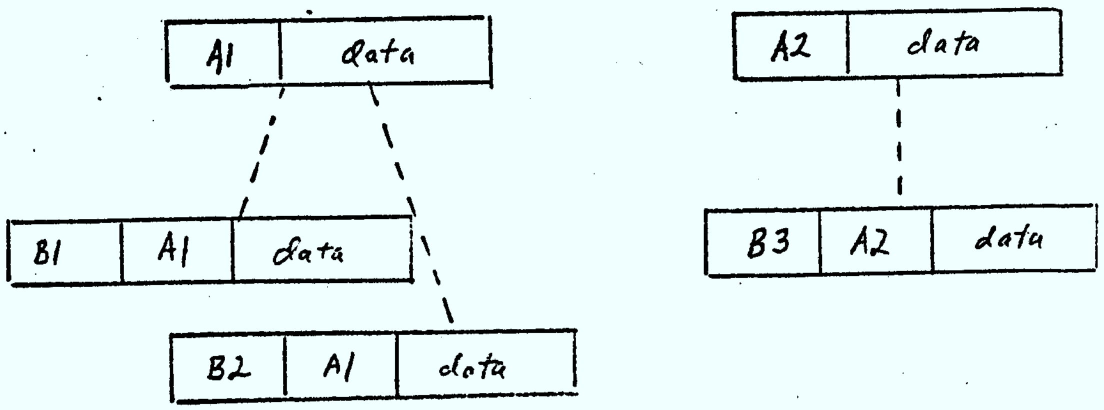
It should be noted that only "information-carrying" associations need be implemented with links for data bases physically stored on RAP (10). All other rela-



a) 1:N Associations for Hierarchy or Network



b) RAP Relation Format



c) Example Record Occurrences

FIGURE 8 - IMLEMENTING 1:N ASSOCIATIONS WITH RAP RELATIONS

tionships which can be derived directly from the values in the records can be handled through the cross marking selection instructions.

Because RAP can be programmed to support multiple user views it can be considered as the basis for implementing a conceptual schema (17). This is a mechanism that provides an intermediate normal form or common denominator for the supporting of multiple external views of the same data base (17). That is, we propose that the RAP data model and instruction set should be a viable candidate to implement such a data base system. A query from any one of the user views can be translated into RAP programs manipulating RAP relations structured to reflect necessary associations.

4.3 A Note on Protection, Security and Integrity

The most general approaches to security and integrity assurance in a data base system are through modification of the original query or generation of internal queries to check the validity of the operation (17). These techniques are highly compatible with RAP's query oriented instruction set. In addition, the qualifications added by query modification will often not affect the execution time on a RAP processor. This happens because the RAP qualification logic is executed on the entire RAP contents and a qualification can have one or several conditions to check often without affecting performance.

5. RAP AND COMMUNICATIONS

5.1 Communications Problems

A crucial component in the origin of the RAP system concept and its subsequent implimentations was consideration of rational data communication. The failure of alternative data base machines and systems is largely a failure to recognize the nature of data communication implied by DBMS. The result is that existing data base machines and system of machines suffer from inherent bottlenecks to which the bandaids of more computing power and increased external communication have been applied.

The real problem is very basic. It is that there has been too little examination of the root causes for the vast transfer of data within computing machines, between machines and peripherals, and from machine to machine whether near or far. Upon some examination one realizes that one underlying cause is a pathological response the inherent lack of precision in human communication. When one is not sure what he wants in detail, he usually asks for everything. Vast piles of unreadable computer printout attest to this tendency. These piles are of course also a commentary on the need for assurance and security, more of which later.

The pathology of uncertainty is, unfortunately, regenerative: For example, the more one needs paper printout, the more one needs very high speed printers; the more spectacular is the printer, the more likely is the motive for a batch system; the more remote the mechanism for computer response, the stronger is the open-ended needs for data less and less precisely specified. All this is understood in detail, but not in its generality: Interactive terminals are an answer to the need for immediacy, a counter to a batch philosophy, an alternative to printouts; but their impact is superficial; the next layer of the onion is still the same; it is fraught with communications problems.

5.2 The Inherent Conflict Between DBMS and General-Purpose Computing

From its very beginning the general purpose digital computer has possessed a property which is, at once, its strength and its weakness. It is this property which has shaped computing machines at every stage of the evolution of hardware and software into what is essentially the same mold - that postulated by Von Newmann three decades ago. The property is this:

The general purpose computer is a crutch upon which one rests, having identified the existence of a problem, while one perceives the direction of its solution, confident that the genrality of the crutch will also assist in one's passage to its resolution.

But, just as the crutch benefits the crippled, so also it can impede the fit. The very generality of the conventional computer makes it less than perfect - overkill at best for any specific task. All of its special features, whether hardware of software, are in support of the nonspecificity of the unknown. Since it is unknow what feature is more important, all are created equal. Since total parallism of their actions is prohibitively expensive, each is available, but only one at a time.

The resulting processor, while more and more general, remains largely serial at its micro and macro levels. It is still Von Neuman's original machine with token parallism, replicated features etc. It still does one of very many things, one at a time. Incidentally, it is the reluctance to recognize the intrinsic inhibition brought by generality which explains the general lack of success of computer science in its search for methodology and structure for parallel processing.

One may now realize the real problem faced by current data base systems. They are, lacking anything else, implimented on general-purpose machines. While DBMS problems are inherently parallel, general-purpose machines, for economic generality, are really serial. If serial they must be faster and faster to do bigger and bigger (general-purpose) jobs. As they became internally faster, they require more and more appartus for bandwidth matching to the inherently slower bulk storage and external communications paths. Bandwidth, limited by the speed of light, demands parallism of connectism at the wire as well as the function level. The numbers of required input/output processors increase. Software support for this generality burgeons; but it is all undirected; it is general purpose, it impliments what might be called the 370/ ∞ solution.

5.3 RAP and Communications

RAP, in the barest sense, is based on a recogntion of the communications problems inherent within a conventional general purpose processor (GPP) running a DBMS. Though the problems are many and interacting, three only will emphasized. They are that conventional general purpose computers are essentially serial in operation, explicit in addressing, and that the storage implied by a DBMS is vast and accordingly slow for reasonable cost. Incidentally the latter description, "vast", is not an absolute one. Compared to the storage required for other purposes in the typical DMBS system of whatever absolute size, or cost the space for storage of data is very great indeed.

The serial nature of a GPP demands very high speed operation to handle a (typically (relatively)) large DBMS data base. Though obvious if one "naively" considers exhaustive search as the access mechanism other "more knowledgeable" approaches founder on the same basis. The data is often inherently poorly structured for more than a single query type. Yet it must be stored in systems with inherent location structure. As query scope expands, more and more complex data sturctures, dictionaries, access paths or whatever are required. The overhead for the storage and dynamic update of these mechanisms grows atrociously with the generality of the system. The overhead of upkeep and the complexity of list search again imply very hight speed operation of an inherently serial processor.

Resulting general-purpose systems become very badly balanced from a bandwidth point of view. Circuits are genrally running with speeds limited by interconnect propogation time, since in the range of the last decade of logic circuit speeds, power per logic operation increases sharply with a resulting decrease in convenient logic density. The usual solution is microparallelism: more wires larger words, wider adders etc. However, economics dictates use of this technique close within the processors, leading in turn to close-packed processors/channel architectures.

When distances become large, as they are to bulk storage devices, economy of interconnection tends to reduce the space bandwidth of the interconnection. The geometry of the situation encourages mini-parallelism in which very many storage devices couple to many controllers, to several channels, to one or two or so cpu's. There is a parallelsim, but it is on a small scale and focussed to centrality. The result is an enormous total bandwidth (in both frequency and space) concentration at the centre. The channels on the GPP grow without bound, driven by a geometry-based square law effect.

On the other hand one should recognize that in the bulk storage devices bandwidths are modest. There is probably a law that says they should always be so, for if the technology of low cost bulk storage supported high bandwidths then it would become the centralized high speed technology and so on for another cycle of lower cost bulk stores..

Of course, they key to the RAP system is the recognition of these facts of communication: If the data is physically dispersed, available at low bandwidths, inherently homogeneous and potentially always available as it is in cyclic stores such as discs and their replacements, then one should process it in place. In 1975, at the beginning of RAP, the possibilities of data compression, both for interrogation and response using the RAP concept, seemed enormous. Subsequent simulation and measuremtn on pnysical models showed this to be so as the attached documents attest.

That the arguments leading to RAP are even more general and have more global application may be obvious at this point. However let us expand upon this possibility.

5.4 Data Base Processors at the Nodes of a Communications Network

5.4.1 A Societal Problem

Of the many large credibility gaps which appear to surround us, there is none as large or far reaching as that separating the realtiy of data base systems from its perception by society. What is most crucial is that this gap is apparently recognized by few and admitted by fewer. Yet it is in the area of data base systems that the average citizen, manager or politician sees both the power and threat of computers. Yet these beliefs are founded on hearsay and conjecture supported by the self-serving implications by computer manufacturers and professionals.

The generality of and economy of query and response from data systems, as viewed by the ultimate citizen user, is several orders of magnitude separate from what is available, or likely to be available, with existing hardware and software technology or simple extrapolations of what is now provided. Unfortunately, far-reaching decisions are made a simplistic basis. The U.S. freedom of

* GPP General Purpose Processor

information legislation for example was encouraged in part by this simplistic view of what is known in an organized way. Others will copy this lead on the same basis. Yet the simple questions that can be asked, yet are not answerable at reasonable cost are legend. Most seriously the cost is not simply that enormous one of using inadequate systems of people and machines to provide an answer but the social one which delay in response will exact in reinforcing societal cynicism.

5.4.2 A Technical Problem

The reality of the service provided by a data base system is that the more you know, the better. As the intellectual scope of the data base decreases it is possible to do more and more to reduce the cost of response to queries. If every possible query type is defined, then the system can be tuned to give economical response somewhat independent of the size of the data base involved. Thus an airline reservations system can operate effectively in its constrained domain of routes, flights and bodies with neames, but would be very hard pressed to answer a query on the age distribution of its passengers by route. It is simply not set up to do this nor would it be expected to be, except as a very, very background job that may be possible at relatively great expense.

Contrast this with a more public service, say a legal repository residing at the node of a computer network. Obviously a simple index of decision by single crime type is trivial, but what about queries involving many attributes, say a, b and c. First of all these attributes must have been identified originally when the index was prepared. But what was once noteworthy and subject to indexing no doubt changes. As the system expands and is used more and more, additional classification will doubtless be needed.

The difficulty is that the process of tuning for classification is far from linear. Generally as each new attribute is tuned all others suffer. One might argue that faster computers are the solution, but public demand and aspiration will normally outstrip the public's ability to pay or technology to provide.

In such a system two levels of procedure will predominate. The first, at the simple level of tuned query and response, will require a fairly brief exchange with a user at a remote terminal, the time being dominated likely by the processor activity. Since the query system is relatively unsophisticated a fairly large amount of data will usually result. However some queries will be unanswerable in real time. Some of these answers may be available overnight at increased cost. Some will be available days later after special programs are written at much increased cost. Some will not be available at all. Probably the alternative of overnight shipment of a large part of data base will be offered. In this case the local enquirer can cull the data by hand or machine on his own time scale as needed.

In summary the communication between a user at a terminal and a data base resource node in a conventional system amy be characterized by

a) Short people orginінated rquests into a node.
b) Modest delay in response.
c) Long computer generated resonses from a node.
d) Occasional delayed, batched, mass transfers of large amounts of data from a node.

5.4.3 A Total Solution

If the node were a RAP machine or RAP machine-assisted, there would be little or no query tuning. Access would be largely by association with no particular knwoledge of the system or its content required. There would be no universal incremental cost of change in user aspiration except for general busyness at the node. There would be non-linear "blocking" effects. Queries would be interactive, of the class "Do you have anything about x?" with a simple quantity and request for detail in response. There would be very few requests that are unanswerable with reasonable delay and an automatic means to induce refinement of the request to minimize time. There is almost no need for mass shipment of data for local perusal. Responses generally would be short, focussed and quantitative, except those like "list all people" which would be refused or sent by mail (heaven forbid!).

In summary the communication between a user at a terminal and a data base resource node with a RAP system may be characterized by

a) Very short people-originated rquests into a node with repetative iteration.
b) Reduced delay in response for the average query.
c) Short computer generated responses from the node interspersed with a).
d) Almost no mass, batched data transfer except as well-formed reports with high utility originating from a node.

5.5 RAP in a Communications Environment

5.5.1 Remote Data Bases

Communication from a remote site to a RAP supported data base can be either in the nigh level language SEQUEL or in RAP assembler. Either form provides a relatively tight and efficient scheme whose use would depend somewhat on the application and the nature of the originating termainal. The query/response process would likely be extremely interactive, though it need not be. The system suits a hierarchy of users of increasing sophistication in an effective way. Provided a great deal is known of existence and nature of the queried data very tight and effective queries can be written straightforwardly. If nothing is known, even the existence of data, then the process is inherently selfdirecting (but see Security, later).

It is natural to form queries to provide statistics of the rquired response while transmitting virtually no explicit data. Thus an interactive user can comprehend the nature of his request without generating enormous system overhead. In the event that an inadvertent request for vast amounts of data is given it is relatively straightforward for a RAP system to respond with statistics and a rquest for clarification.

In summary communication with a remotely located RAP system can be nearly ideal requring only a few bytes of relatively unsophisticated query with a few bytes response in an interactive dialog which is inherent in the processor itself. More sophisticated users can expand their requests enormously with far less than proportional effort while at the same time retrieving extremely explicit results requiring little communications overhead and in relatively short time.

5.5.2 Distributed Data Bases

The RAP scheme is ideally suited to distributed dynaimic data bases: Because RAP intrinsically minimizes structure in the data, update is not traumatizingly long. Thus the period of lockout from general queries required by a conventional system to ensure that a response is not based on some unknown combination of old and new data, is very small, of the order of "revolution".

RAP systems can likely be easily arranged in parallel or hierarchically such that requests for existence can be "broadcast" to all participants. Responses to a "broadcast" query can be used to focus more normal RAP machine/user dialog. An exception is the case in which attributes of a particular query subject are spread geographically. In this case, naturally, various cell to cell interactions, such as cross-mark, are not possible in the usual way. Such operations would generally require much greater exchange of data between sites. The possibility however exists in a RAP network of merging files temporarily at an intermediate geographic site for detialed query execution.

An interesting special case is one in which each user has a mini-RAP as part of his query termina. Normally his interaction is with his own data base stored locally. A retail store inventory system might be a simple example. Upon the occasion of seeking stock elsewhere to fill an immediate customer demand access to other RAP machines would be straightforward. The interaction would likely be interactive as the optionality of the product, its physical location etc. etc. suited the customers requirements. Various centralized inventory, product history, sales prifles etc. data could easily be acquired by a head office connected to such a system.

5.5.3 Privacy and Security

Security of information with a RAP system is likely to be superior to that with any other approach. This statement appears to be in conflict with the "fishing" attribute of RAP systems which generally allows queries based on no real fact; but it is not.

RAP's ultimate strength is that its data is as "unfied" as one wishes or needs. Since the data is its own key it can be arranged that all of it exist at one place, in one "file", with no other vestigest. Specifically there need not be dictionaries, lists, access paths etc. which hint at the existence of data. In conventional systems, protection of such lists is extremely comples, since the access path approach is already ponderous with duplication and redundant data. To add simple privacy codes might be easy but not adequate. A real situation is much more complex, with a dynamic and data dependent "need to know" system normally required.

The real advantage of RAP, and particularly distributed RAP, with respect to privacy is the possibility of maintaining keys with all of the data which can be used to implement a thorough protection system including query trace.

The latter is probably useful to follow up certain pathological requests for "statistics" which can inadvertently provide information which is masked for explicit use at a higher level than that at which its occurrence can be counted.

One of the potentially most powerful techniques available, particularly with distributed RAP or mini RAP buffered queries, is the possibility of keys which trigger reverse queries and/or trace recording.

6. ADKNOWLEDGEMENTS

6. ACKNOWLEDGEMENTS

This research was funded by the Department of Communications, Department of Supply and Services, and the National Research Council of Canada. We gratefully, Thank Intel Corporation for their donation of CCD and associated driving components. We would also like to acknowledge and thank the members of the RAP project that have contributed so much to its accomplishments: M. Chan, A. Cousin, R. Freen, C. Hawkins, R. Hudyma, H. Huwito, W. Lane, R. Nakaňo, B. Patkau, P. Pereira, A. Radaez, R. Reid, P. Sadowski, M. Soong, K. Sevcik, J. Klebanoff and A. Tsonis. A special thanks to M. Soong whose memo on the RAP. 2 instruction set was used as a basis for section 2.

7. REFERENCES

7. REFERENCES

1. Ozkarahan, E.A. Schuster, S.A. and Smith, K.C, "A Data Base Processor" Technical Report CSRG-43, Computer Systems Research Group, University of Toronto, September 1974.

2. Ozkarahan, E.A., Schuster, S.A. and Smith, K.C., "RAP--An Associative Processor for Data Base Management", Proc. AFIPS NCC, vol. 44, 1975, pp. 379-87.

3. Schuster, S.A., Ozkarahan, E.A., Smith, K.C., "A Virtual Memory System for a Relational Associative Processor", Proc. AFIPS NCC, Vol. 45, 1976, p.p. 855-862.

4. Oskarakan, E.A., "An Associative Processor for Relational Data Bases-RAP", PH. D. Thesis, University of Toronto, 1976.

5. Kerschberg, L., Ozkarahan, E. A., and Pacheco. J.E.S., "A Synthetic English Query Language for a Relational Associative Processor", Proc. of the Second International Conference on Software Engineering, October, 1976.

6. Ozkarahan, E.A. and Schuster, S.A., "A High Level Machine-Oriented Assembler Language for a Data Base Machine", Technical Report, CSRG-75, Computer Systems Research Group, University of Toronti, October 1976.

7. Nakano, R. "A Simulator for a RAP Virtual Memory System", M.S. thesis, University of Toronto, 1976.

8. Ozkarahan, E.A., Schuster, S.A., and Sevcik, K.C., "Performance Evaluation of a Relational Associative Processor", ACM TODS Vol. 2, No., June, 1977, pp. 175-195

9. Ozkarahan, E.A. and Sevcik, K.C., "Analysis of Architectual Features for Enhancing the Performance of a Data Base Machine", (in press), ACM TODS.

10. Sibley, E.H. (Ed), "Special Issue: Data-Base Management Systems", ACM Computing Survey, Vol. 8, No. 1, June, 1976.

11. Lowenthal, E.I., The Backend Computer, Auerbach, 1976.

12. Copeland, G.P., Lipovski, G.J., Su, S.Y.W., "The Architecture of CASSM: A cellular System for Non-numeric Processing", Proc. First Annual Symposium on Computer Architecture, 1973.

13. DeFiore, C.F., and Berra, P.B., "A Data Management System Utilizing an Associative Memory", Proc. AFIPS NCC, Vol. 42, 1973.

14. Lin, C.S., Smith, D.C.P., and Smith, J.M., "The Design of a Rotating Associative Array Processor for a Relational Data Base Management Application", ACM TODS, Vol. 1, No. 1, March, 1976, pp. 53-65.

15. Baum, R.I., and Hsiao, D.K., "Data Base Computers -- A Step Towards Data Utilities, IEEE-TC, Dec., 1976, Vol C-25, No. 12.

16. Zaky, S.G., "Microprocessors for Non-Numeric Processing", Proceedings of Third Workshop on Computer Architecture for Non-Numeric Processing, SIGARCH Vol. VI, No. 2, SIGIR Vol. XII No. 1, SIGMOD Vol. IX No. 2, May, 1977, pp. 23-30.

17. Tsichritzis, D.C., and Lochovsky, F.H., Data Base Management Systems, Academic Press. 1977.

8. DESIGN/PERFORMANCE APPENDICES

8.1 RAP 2 - Cell Logic Design

8.1.1 General

The design of each of the two cells implimented is given in logic diagrams L-1 to L-19 appended in section 8.1.4. Each cell, with 320 kilobits of memory requires, 412 IC packages occupying 13 boards. A memory expansion on each cell to 1.5 megabits uses 128 IC packages on 4 boards. Each cell exclusive of memory uses 9 amps at 5 volts for .45 watts power dissipation.

The logic design was done with little attempt at minimization of any paticular sort. Rather, flexibility for options and improvement was sought if time permitted. Design is largely small and medium scale integrated T^2L with Schottky used where necessary. Each cell employs 218 SSI, 117 MSI, and 77 LSI IC packages with 96 IC's required for each megabit of storage.

The remainder of this section consists of a functional description of each logic diagram in turn.

8.1.2 FUNCTIONAL DESCRIPTION

8.1.2.1 Logic Diagram L-1

1. The CONTROLLER communicates with the cells via the RAP CONTROL BUS and RAP DATA BUS. Each cell must have its own buffer inverters to reduce the load seen by the CONTROLLER.
2. The values set on the RAP CONTROL BUS are decoded by several decoders which are typically prohibited/enabled by various states of the cell so that the CONTROLLER can restrict communication to a few selected cells. KEY_ENABLE is used to prevent any switching noise on the CONTROL BUS from being interpreted as a valid command. RUNNING is also used to prohibit some decoders in order to protect the contents of its registers while a cell is scanning its track. Each output of a decoder corresponds to a task; for a full description of these tasks, see Appendix B.

8.1.2.2 Logic Diagram L-2

This diagram shows the logic of all important states of the cell.
1. ADDRESS SWITCHES are a set of 8 SPST switches settable by a human operator. Their "contents" are readable by the CONTROLLER (See L-13) and are compared with the RAP DATA BUS during an attempt to open the cell by address (by CK 1).
2. The RELATION NAME REGISTER (abbr: RN REG) contains a 16-bit pattern which is a coded name of the relation name assigned to the cell. Its contents can be set (by PLD 1) and read by the CONTROLLER (See L-13). They are also compared with the RAP DATA BUS during an attempt to open the cell by relation name (by CK 0). Thus, the relation name is treated as a programmable address.
3. There is a provision (by CK 4) for opening all cells independently of their state , address and name. This is not essential but it helps to accelerate the initialization process after power is turned on.
4. When the cell is "destroyed" (by PLD 9), ALIVE=0 and the RN COMPARATOR is negated; thus the cell cannot be opened by any relation name until a new name is assigned to it (by PLD 1) when ALIVE becomes high.
5. Each cell must be initiated to run; i.e. RUNNING is set high by either CK 6 (for all cells) or by PLD 8 (for some selected cells). After a cell has been running (scanning its track), there are very few things the CONTROLLER can do to it; it is therefore important to load all necessary constants before initiating a cell to run.
6. A cell can be made to terminate its activities and erase most parameters of the "current" instruction when EXFIN is generated (by CK 7 for all cells

unconditionally or by INS 4 for some selected cells). Most of the time, this is not essential but it can help to simplify the process of loading parameters of the next instruction. The only circumstance for which it must be used is after an instruction with a Query List involving a test on the address (See L-7).

7. POLL_IN and POLL_OUT belong to the single line that runs through all cells and that defines priority by physical location. The line is used with the GET_NEXT_CELL scheme (i.e. with INS 3) to select exactly one cell and to "advance" the selection along the line. This unique selection is important in the reading of most information (but STATUS) from a cell and in the insertion of new tuples. REJECTED indicates that a cell has been served and BLOCKED means that it must wait until its turn. Note that only open cells are affected by this selection scheme; others are totally "transparent" to it.

This scheme (and the priority line) is not essential since one can always select by address. However, for a large system, this scheme can help to reduce both time and space.

8.1.2.3 <u>Logic Diagrmas L-3</u>

1. BLOCK_START is a narrow pulse marking the beginning of every tuple while the cell is running (See L-17). It is stretched until the next falling edge of Φ to become BS which initiates the generation of $\phi1$ thru $\phi7$ corresponding to the first seven bit-times of every tuple. SQ1...SQ7 are shorter versions of $\phi1...\phi7$.

2. GAPE is low while a tuple is being scanned. It is found to go up again at the end of every tuple by DL.

3. Before (as well as after)every domain exist two bits that were employed in RAP I to store the length code of the next domain.LC1 and LC2 are the bit-time signals for these bits. $\phi6$ and $\phi7$ provide these bits forthe first domain and, as mentioned earlier, their generation is due to BS. PLC1 and PLC2 provide these bits of all subsequent domains; they are generated by the termination (MOR) of every domain. In RAP 2 , the length code bits are no longer necessary since the LENGTH CODE RAM stores all format information . It is difficult, however, to remove these bits without extensive changes in timing. These bits and the corresponding timing signals are therefore maintained but they no longer carry any information except that the first two are still reserved for the TKE symbol.

4. The WORD LENGTH COUNTER is (synchronously) preloaded with a proper value which depends on the length of the next domain. When a domain is being scanned, EOR allows the counter to operate; after the clock pulse of the last bit of the domain, the counter becomes full (indicated by its TC terminal), MOR is generated to mark the end of the domain and to initiate the generation of PLC1 and PLC2.

5. The information on the mark-bits are sampled by SQ1 through SQ5 to get DF, A, B, C and D.

8.1.2.4 Logic Diagram L-4

1. WORD POSITION COUNTER (abbr: WPC) is the pointer indicating the next domain (the 1st domain of any tuple is numbered zero) relative to the "current" tuple. When the cell is running, it is incremented by EOR which is a signal covering every domain. In a CREATE process, while the CONTROLLER is loading format informations into the LENGTH CODE RAM, the counter is incremented automatically. It is however the duty of the CONTROLLER to reset (with PLD 11) the counter to zero before starting to load. When the cell is initiated to run, the counter is automatically reset (by EXRUN).
2. LENGTH CODE RAM (abbr: LCR) stores the length codes of all domains: Since there is only one format per cell, the RAM has to be only deep enough to accommodate the maximum number of domains per tuple. In RAP II, this number is 255. For "safety" reasons, the LCR can be loaded only while the op-code for CREATE is present. KEY_ENABLE must last long enough to write on the RAM.
3. Since the WPC always points to the next domain, the length codes of the current domain must be pre-sampled by LC1. They are then decoded to see if the READ HEAD is now at the end of a tuple (contents of LCR is "11"). In this case, DL is generated. Otherwise, there will be another domain coming next and EOR is generated slightly before the 1st clock pulse of that domain, and one of E10, E18, E34 is also produced to control the WORD LENGTH COUNTER and to define the proper length of many shift registers. EOR is reset at the end of every domain by MOR.
4. Due to an historical reason which was thought unimportant to change, the length code bits of the 1st domain of every tuple is used to store the "logical track end" symbol (TKE). If both these bits are marked, it means that there is no more valid data beyond this tuple. TKE is therefore generated from the sampled values of DATA_IN at $\phi 6$ and $\phi 7$. In the case of data retrieval, when enough satisfied tuples have been taken, a false TKE condition is "faked" so that the cell stops properly. TSYN assures that TKE is reset before any instruction is carried out.

8.1.2.5 Logic Diagram L-5

This diagram shows the production of EQUALITY and EQUALITY2 which are employed to cover the operand(s) in an arithmetic instruction or the domains to be read out in a partial retrieval instruction.

1. A specified domain-number must be preloaded by the CONTROLLER into one of three 8-bit registers where it is compared with the WPC to generate either

EQUALITY or EQUALITY2. Typically, only EQUALITY is generated (at most 3 times per tuple). EQUALITY2 appears only for the instructions ADD, SUB and REP provided INT-1 ≠ INT-2 (See L-11 and L-12).

2. The domain number corresponding to EQUALITY2 must be loaded into the middle 8-bit register. Y1 prevents EQUALITY2 from being generated by the contents of the other two registers.

3. In an arithmetic instruction (indicated by C_op4=1), EQUALITY corresponds to the leftmost register. Y0 prohibits the other two registers from giving any interference.

4. For historical reasons, the three 8-bit registers share the same comparator. Therefore, a restriction must be made as follows:

- The leftmost register has the highest priority; the rightmost the lowest (EX: If only one domain has to be covered, the leftmost register must be used).
- A smaller domain number must be stored in a higher priority register.

To simplify software/hardware, the three registers also share the same loading signal (SLD 4). The smaller domain-number must be shifted in later. It is also important (in data retrieval only) that unused registers must contain zero so that unwanted matching with the comparator cannot occur.

5. Since the three 8-bit registers are connected one after another through tri-state gates to the comparator, it is important that there always be one register remaining connected (otherwise a false match could result). For this purpose, the signal Y2 is used to prohibit the 2-bit counter from being incremented.

8.1.2.6 Logic Diagram L-6

There are three independent Domain Comparator Units that are virtually identical; therefore only the first one is shown in L6 and analyzed. Each unit is used to compare a domain with some constant as specified by the Query List.

1. To use the first unit, SL-4 must be preloaded with a value of "1". The values of f1, g1 and h1 must also be specified. The comparand must be loaded (by SLD 5) together with the domain-number of the domain to be tested. There are four shift registers for this purpose. The leftmost stores the domain number (which must be loaded last). The next one is for an 8-bit comparand; if it is 16-bit, one more register is used. Finally, for a 32-bit comparand, the rightmost register stores its least significant part.

2. In operation, PQ1 is generated (in a similar way to EQUALITY) to cover the specified domain when the contents of the leftmost shift register match the WPC.

PQ1 allows the clock Φ to be used (as ϕc1) for the comparison. ϕc1 rotates the comparand and samples the "current" result of the comparison in two J-K flip flops. The comparison of the comparand with the value of the specified domain is made through two EX-OR gates and a few other gates under the assumption that numbers are in pure binary. A D-flip flop storing the sign difference is used with two more EX-OR gates to accommodate 2's complement notation. The final results are gated with f1, g1, h1 and then sampled by PLC1 (first bit after the last bit of the relevant domain) to produce Q1. If the comparison is satisfactory, Q1 is high after the domain is scanned and stays high until the next tuple; otherwise Q1 will be low.

8.1.2.7 <u>Logic Diagram L-7</u>

This diagram shows how the mark-bits and the result of the Domain Comparator Units are evaluated to produce the QDCJ signal. QDCJ means "qualified either disjunctively or conjunctively"; that is to say the relevant tuple satisfies the Query List.

1. DISJ is a value loaded by the CONTROLLER. It must have the value of "1" if the Query List is in the disjunctive form; zero otherwise.
2. QMRK (= "qualified as far as marks are concerned") is high for a tuple if:
   - at least one specified mark-bit has a mark.
   - Furthermore, if DISJ=0, all specified mark-bits must have mark.
3. QUMRK (= "qualified as far as absence of marks are concerned") is similar to QMRK except that it concerns the absence of marks on mark-bits.
4. QDCJ for a valid tuple is high after all necessary information has been acquired if:
   - either no item in the Query List is satisfactory
   - or (in the disjunctive mode only) at least one item is satisfactory.

   One simple way to select no tuple is to set DISJ=0, a2=a3=1. This asks for the impossible case that the mark-bit A has and has not a mark.

   In the case of "LOAD I/O BUFFER", after enough tuples have been read, the cell is forced to scan one more tuple; for this extra tuple, QDCJ is forced low by a (possibly faked) TKE condition.
5. ADR is automatically set when a comparison against the ADDRESS SWITCHES is made for the execution of the Query List (ADR is not affected by the opening of a cell by address). Many such comparisons can be made for the Query List and just one match is enough to set HMA high. It is important that EXFIN is issued after the CONTROLLER has finished with an instruction involving address test.

Otherwise , the value of ADR and HMA can interfere with any future query test. The concept of address test is necessary only in RAP I and totally undesirable in RAP II; it is implemented here just for the sake of keeping the original RAP specifications unchanged.

8.1.2.8 Logic Diagram L-8

1. Most instructions demand action if and only if a tuple satisfies a Query List. These instructions are characterized by the fact that they involve only valid data and can be terminated by a TKE condition. The rest, namely SPACE COUNT, INSERT and CREATE, are quite different: They involve non valid data (i.e. vacant space) and consequently need no Query List and cannot be terminated by a TKE condition.

2. The signal SATISFIED is used to indicate that some actions must be taken for a relevant tuple. For the reasons discussed above, most instructions must have their SATISFIED signal generated by sampling the value of QDCJ by DL; whereas INSERT and SPACE COUNT by sampling the value of DF by SQ1. CREATE is quite different since it does not depend on the data on the track; but rather it seeks to see if there is enough space to place a future tuple. Consequently, in CREATE, the signal SATISFIED must be generated by DL which marks the end of a tuple space.

3. The S-COUNTER counts the number of satisfied tuples in a pass. Its contents are compared with a limit loaded by the CONTROLLER in the LIMIT REGISTER for the instructions INSERT and LOAD I/O BUFFER. After the limit is reached, no additional SATISFIED can be produced (and hence there is no more action) and the S-COUNTER is frozen. In the case of INSERT, the cell stops after inserting the last tuple required. In LOAD I/O BUFFER, the READ HEAD still scans one more tuple after having read the last satisfied tuple required so that the WRITE HEAD can have a chance to mark/unmark properly. Since the extra tuple can be satisfied and can cause a problem (see L-10), it is necessary to fake a TKE condition for that tuple.

4. RHOFF indicates that the READ HEAD has gone beyond the physical end of the track and obviously there are no other satisfied tuples to be counted.

5. WHOFF indicates that the WRITE HEAD has gone beyond the physical end of the track and obviously there is no more space to write on and the cell must stop.

6. TABLE 1 shows the timing of the signal SATISFIED

TABLE 2 shows the timing of the signal AUTOFIN.

7. Note that in RAP I, there must be at least one TKE tuple left on the track. In RAP II, there is no need for this; furthermore, the odd track remnant that may exist (if the length of a block does not divide the track length) cannot cause any problem.

TABLE 1

Properties of the signal SATISFIED

	INSERT	SPACE COUNT	CREATE	LOAD I/O BUFFER	OTHERS
Criteria of generation	DF = 1	DF = 1	All tuples	QDCJ = 1	QDCJ=1
Prohibited when the S-COUNTER reaches a predetermined limit	Yes	No	No	Yes	No
Timing	Immediate	Immediate	Delayed	Delayed	Delayed

Immediate:	from SQ1 of relevant tuple until DL .
Delayed :	from DL of relevant tuple until next DL.

TABLE 2

Generation of the signal AUTOFIN.
If many conditions can cause AUTOFIN, only the one which comes earliest will.

Moment of generation	INSERT	SPACE COUNT	CREATE	LOAD I/O BUFFER	OTHERS
WHOFF = 1	Yes	Yes	Yes	Yes	Yes
Last bit of the last required tuple	Yes	N/A	N/A	No	N/A
Last bit of the tuple beyond the last required tuple	No	N/A	N/A	Yes	N/A
Last bit of a tuple bearing the symbol TKE	No	No	N/A	Yes	Yes

8.1.2.9 Logic Diagram L-9

1. REN is the signal that allows writing to be performed for the CCD track and TTDATA is the data to be written there when REN is high (See L-14 and L-18).
2. For INSERT, a vacant tuple is completely erased (including all mark-bits) and rewritten with the serialized contents (INS_DATA) of the I/O BUFFER; the option MARK can be used to preset (with SBIT) all new tuples.
3. For DELETE, the whole tuple is also erased; but the 1st mark-bit is marked by SQ1 through SBIT.
4. For CREATE, the whole track is erased and marked with DF and TKE symbols (i.e. the 1st, 6th and 7th bits) by SQ1, SQ6 and SQ7.
5. By examining carefully diagrams L-8 and L-9, one will find that the remnant that may exist at the track end will pose no problem: It will be erased and it is never misinterpreted as a tuple.
6. For most instructions but SPACE_COUNT, the MARK/UNMARK option will cause the marking/erasing of the mark-bits (specified by a1, b1, c1 and d1) by the turning on of REN at the right time. MARK overrides UNMARK since SBIT is interpreted as data to be written. It does not make sense (though permissible) to use the MARK option for DELETE and CREATE.
7. For some arithmetic instructions (more specifically, for ADD, SUB and REP), the result to be rewritten on the track is expressed by OR1P; and UPDATE covers the domain to be updated.
8. Except for the above rules, the contents of the track cannot be changed by the cell in any other circumstance. It is possible however to change manually (for testing purpose) as described in L-14 and L-18.

8.1.2.10 Logic Diagram L-10

1. The I/O BUFFER serves only to implement the insertion/retrieval instructions (INSERT and LOAD I/O BUFFER). As its name implies, it serves as a buffer to decouple the speed of the CONTROLLER (or other external devices) from that of the cell.
2. In INSERT, the CONTROLLER loads as much data as it wants; limited only by the size of the I/O BUFFER (which is currently 1Kx16 RAM). A 16-bit domain fits nicely in one RAM word. An 8-bit domain must be loaded into the lower byte of the RAM. A 32-bit domain must be loaded in two consecutive RAM words; with the least significant half first. The CONTROLLER also has to load the number of tuples to be inserted into the LIMIT REGISTER.

The BUFFER has its own pointer (I/O BUFFER COUNTER) which can be reset to zero by the CONTROLLER (using PLD11). Every time a word is loaded (by SLD1), the signal BT is generated by a one-shot to transfer data from the RAP DATA BUS into the RAM. Note that the op-code for INSERT must have been preloaded before this transfer can be done correctly. A second one-shot is used to advance the pointer.

In operation, the pointer is reset to zero by the initiating signal EXRUN. When a vacant place is found (SATISFIED is set by DF), a BT pulse is generated before the first clock pulse of every domain to transfer the "current word" into a parallel-to-serial converter. For a 32-bit domain, after the first 16 bits of the domain, a second BT pulse is generated (this is done by decoding the contents of the WORD LENGTH COUNTER) to get the second half of the domain. The contents of the parallel/serial converter are shifted out and go through a D-flip flop where they are chopped by RSQ into a stream of pulses named INS_DATA (absence of a pulse indicates a zero) which will be sent to the UPDATE CONTROL unit (L-9) for writing onto the track.

3. In LOAD I/O BUFFER, data from the track is continuously shifted into a serial-to-parallel converter and a BT pulse is generated immediately either after the last bit of every domain or after its 16th bit to write the converter's contents into the BUFFER. Also, the pointer is incremented accordingly. This is done for all valid tuples even non satisfied ones. Since we do not want to store non-qualified data, we must overwrite it with new . In other words, we must backtrack if a tuple is found to be not satisfied. Naturally, the backtracking is not done on the track but by resetting the pointer to a proper value. This is possible thanks to a (not named) register which remembers (by SQ5) the value of the pointer before any tuple is read in. At the end of the tuple (marked by DL), if it is not satisfied (QDCJ=0), that value is returned to the pointer.

Finally, in the partial retrieval mode, SL-1=1 and the BT signal is generated only for domains covered by EQUALITY (See L-5) and hence only specified domains will be stored in the BUFFER.

Since the size of the BUFFER is finite, only a limited number of tuples can be read. It is up to the CONTROLLER to find this limit and load it into the LIMIT REGISTER. After enough tuples have been retrieved, the cell will fake a TKE condition and it will stop in one tuple-time after the last retrieved tuple. If there are not enough satisfied tuples, the cell will stop at either the logical end of physical end of the track (See L-8).

8.1.2.11 Logic Diagrams L-11 and L-12

The ARITHMETIC UNIT is employed strictly to support the instructions ADD, SUB, REP, SUM, MAX and MIN which are characterized by C_op4 = 1. Due to limitation of space, we only allow 8-bit and 16-bit operands. It is a straightforward matter to expand the logic to accommodate 32-bit domains; it is mandatory however to define new "keys" to load and read data for a longer field.

1. For MAX and MIN:

These instructions ask for the max/min value of a specified domain of all satisfied tuples to be stored in the RESULT REGISTER. In operation, REGISTER 1 constantly takes the specified domain of all tuples; satisfied or not. Meanwhile, the contents of REGISTER 2 are compared against the specified domain (from the track) withthe help of a serial comparator which employs a few EX-OR gates, J-K flip flop and a D-flip flop for sign correction (note the similarity with the logic of the comparator of the Domain Comparator unit). If TAKE NEW =1, it means that the latest tuple has a "better" value and must replace the "current" max/min value stored in REGISTER 2. However, TAKE NEW is "anded" with SATISFIED so that the result of the comparison is discarded if the latest tuple is not satisfied. Furthermore, for all tuples from the beginning until and including the first satisfied tuples, the serial comparator is deactivated by SAT ONCE=0 and always produces a high value of TAKE NEW. The net result is that at the 1st bit of the 1st tuple after the 1st satisfied tuple, the contents of REGISTER 1 are transferred to REGISTER 2. Briefly speaking, the logic of L-11 assures that the specified domain of the 1st satisfied tuple is always loaded into REGISTER 2 before any more comparison. From now on, the serial comparator is activated (SAT ONCE=1) and any satisfied "better" value will be loaded into REGISTER 2 where it is rotated and compares against the data from the track until a still satisfied and better value replaces it. Furthermore, the finding of a satisfied and better value also triggers an R-S flip flop that allows the generation of CKb which will shift that value (from REGISTER 2) into the RESULT REGISTER. This shifting occurs one tuple-time after that value was first seen. Note that there is no problem even if the max/min value happens to belong to the last tuple on the track since the cell only stops one tuple-time after the track-end. Once in the RESULT REGISTER, the "current" max/min value stays there until it is replaced by a "better" and satisfied one.

After the cell stops the CONTROLLER can read from the RESULT REGISTER using R_ALU (See L-13). Note that if the specified domain is 8-bit long, the result is duplicated in both bytes of the RESULT REGISTER.

2. For ADD, SUB and REP.

These instructions are similar and are characterized by:

- They need two operands and therefore are the only instructions requiring the signal EQUALITY 2.

- They are the only instructions that can cause a change in one domain (of all satisfied tuples).

The need for two operands requires (unless an operand is a constant) the covering of the lower-numbered domain by EQUALITY and that of the higher-numbered domain by EQUALITY2. The relevant domain-numbers must be shifted in by the CONTROLLER in correct order (See L-5). If the 1st domain (lower-numbered) must be updated, the CONTROLLER must preload INT-1=0, INT-2=1. If the 2nd one must be, INT-1=1 and INT-2=0. In the limiting case where both operands are identical. then the 2nd domain-number is ignored since EQUALITY2 will not be generated; the CONTROLLER must set however INT-1 = INT-2 = 1. If one operand is a constant (supplied by the CONTROLLER), only one domain is involved and only EQUALITY but not EQUALITY2 will be generated; the CONTROLLER must set INT-1 = INT-2 = 0. The operation of the cell depends on the values of INT-1 and INT-2:

Case 1: INT-1 = INT-2 = 0 (External operand)

The CONTROLLER must preload the constant into REGISTER 2 using PLD 13 (before the cell is initiated ro tun). If the relevant domain is 8-bit, the constant should reside in the higher byte. In operation, CKa will rotate the constant properly when the specified domain is being scanned. Meanwhile (i.e. while EQUALITY * EOR = 1):

For ADD, the serial full adder adds the output of REGISTER 2 (i.e. the constant) with the current domain value (from track). A D-flip flop takes care of the carry and the result is shifted into the RESULT REGISTER. If the domain is 8-bit long, both bytes of the RESULT REGISTER store the duplicated sum. At the end of a tuple, the sum is already contained in the RESULT REGISTER, whether or not the current tuple is satisfied. While the current sum is shifted in, the previous sum is shifted out through a D-flip flop to become OR1P which is immediately sent to the UPDATE CONTROL UNIT (L-9) to be written on the track. If the previous tuple qualifies, UPDATE is high and the writing process is accomplished; otherwise no writing will be possible (for the previous tuple). In a similar way, the current sum must wait until the next tuple is scanned to be written on the track.

For SUB, the constant is complemented by an EX-OR gate controlled by SUB. Also, the carry flip flop is preset to 1 to complete the 2's complementing of the constant. Everything else remains the same as for ADD.

For REP, the adder is not used and the constant is shifted directly into the RESULT REGISTER. Every other activity is the same as for ADD.

Case 2: INT-1 = 0, INT-2 = 1 (lower-numbered domain is updated)

Both REGISTER 2 and the RESULT REGISTER are shifted while either relevant domain is scanned. The 1st domain is shifted in REGISTER 2 while EQUALITY is high and stays there until it is shifted out when EQUALITY2 is high to be added with the 2nd domain (for ADD) or the latter's 2's complement (for SUB) and the sum is shifted into the RESULT REGISTER. For REP, the 2nd domain is simply shifted directly in the RESULT REGISTER. By the end of the 2nd specified domain, the proper result is stored in the RESULT REGISTER (whether or not the tuple qualifies) where it remains until the 1st specified domain of the next tuple is scanned when it will be shifted out to become OR1P and everything else is similar to Case 1.

<u>Case 3</u>: INT-1 = 1, INT-2 = 0 (higher-numbered domain is updated)

Only REGISTER 2 is shifted while either relevant is scanned. The RESULT REGISTER is shifted only while the 2nd domain is scanned. Like Case 2, the 1st domain is shifted into REGISTER 2 and is shifted out to be added with the 2nd domain (for ADD) while EQUALITY 2 is high. However, if SUB is involved, the contents of REGISTER 2 is complemented before being added. For REP, the contents of REGISTER 2 is also shifted directly into the RESULT REGISTER. Again, like Case 2, by the end of the 2nd specified domain, the proper result is stored in the RESULT REGISTER but this time it must wait until the 2nd specified domain of the next tuple to become OR1P which will be written on the track if the relevant tuple is satisfied.

<u>Case 4</u>: INT-1 = INT-2 = 1 (single operand)

In the case, only EQUALITY is generated and both REGISTER 2 and the RESULT REGISTER will be shifted while EQUALITY is high. If ADD is involved, the value of the specified domain is delayed by one bit-time by a D-flip flop before being shifted into the RESULT REGISTER. The net effect is that the sum is twice the operand. For SUB, nothing (i.e. zero) is shifted into the RESULT REGISTER to form the proper null result. For REP, the operand is shifted into the RESULT REGISTER directly. Just like case 2, the proper result stays in the RESULT REGISTER until the next tuple to be shifted out and written on the track if the relevant tuple qualifies.

3. For SUM

This operation involves only one operand and EQUALITY is used to control the clock for both REGISTER 2 and the RESULT REGISTER.

The specified domain is always shifted into REGISTER 2 (whether or not the tuple qualifies) when it stays until the same domain of the next tuple displaces it. If the relevant tuple is not satisfied, nothing is done; otherwise the output of REGISTER 2 is added with the contents of the RESULT REGISTER. and the sum is stored back in the RESULT REGISTER. The partial sum remains there until a next satisfied tuple is formed.

As mentioned before, there is no problem if the last tuple of the track qualifies since the cell always scans one extra tuple (an "imaginary" one, if necessary) beyond the track end before it stops.

After the cell stops, the CONTROLLER can read the final result from the RESULT REGISTER by R_ALU. If many cells are involved in the SUM instruction, it is up to the CONTROLLER to collect all individual sums and find the final sum.

Note that as long as no satisfied tuple has been found, "zero" is constantly shifted in the RESULT REGISTER. Even if the 1st tuple qualifies, this fact is not known until the end of that tuple; consequently the RESULT REGISTER is always initialized properly to zero.

8.1.2.12 Logic Diagram L-13

This diagram shows clearly everything that the CONTROLLER can read from a cell and it needs little explanation.

1. To read the I/O BUFFER properly, its pointer must be read first to know how many words in the I/O BUFFER are meaningful data. Since the pointer always points to one word beyond the valid data area, if the BUFFER is full, the pointer can point to zero. A check on the "satisfied status" should clarify this.
2. Every time the I/O BUFFER is read, its pointer is automatically incremented by one. But the CONTROLLER must reset the pointer (using PLD 11) before starting to read.

If a domain is 8-bits long, it must be taken from the higher byte. If a domain is 32-bits long, the lower word is read out first.

3. The five least significant bits of STATUS indicate the presence of at least 1 mark on the mark-bits (Ex: If there is at least one vacant place, bit #4 is high). Note however that these bits reflect only the portion of the track seen by the READ HEAD in the last pass; that is to say, after an INSERT or LOAD I/O BUFFER, these status bits do not necessarily show the mark-bits of the whole track. Furthermore, these bits are not affected by whatever was written by the WRITE HEAD. For instance, after a CREATE or after a MARK/UNMARK option, these bits do not reflect what is on the track.
4. Bits 5 to 9 of the STATUS have not been defined. The rest is for important states of the cell. The most significant bit is always high; this is very important to detect the presence/absence of any cell and is necessary as a "feedback" test for the use of "GET NEXT CELL".

8.1.2.13 Logic Diagram L-14

This diagram shows the generation of all important memory phases.

1. All the phases are decoded from the various counts (a, b, c, d) of the master clock MC.

OSC signals the end/beginning of each bit-time. To understand CHOICE, it is necessary to realize that each INTEL 2416 chip is in fact two time-multiplexed drums. CHOICE is used to alternate between them. Φ1, Φ2, Φ3, Φ4 are strictly used for shifting CCD memories.

2. SAMPLE is used mainly to mark the time when data is available from the CCD. Φ is used for many parts of the cell such as shift registers and the WORD LENGTH COUNTER. RSQ is a secondary phase arriving slightly late than Φ to truncate various signals. WE is for enabling the writing process, CE1 is used for enabling the CCD chips.

8.1.2.14 Logic Diagram L-15

This diagram shows how the current position on the track is monitored. It also shows how the single-step mode and the slow mode together with the halt logic are implemented.

1. In the normal mode, the SECTOR COUNTER points to the current position on a minor loop, and the TRACK COUNTER the current minor loop. The SECTOR COUNTER is advanced once every time the TRACK COUNTER is full (LS=1).

2. In the slow mode, which can be entered either manually (by setting a switch properly) or electronically (via EX_SLOW when the cell is running at the highest speed), for each bit-time "given" to the rest of the cell, 256 others bit-times are held back; that is to say the CCD memories are constantly working at the full speed but the rest of the cell "thinks" that the speed is reduced 257 times. In the holding state (characterized by STORE = 1 and bit- times being held back), the TRACK COUNTER is not incremented and the SECTOR COUNTER is used to count the number of held bit-times: At the beginning, the SECTOR REGISTER stores the contents of the SECTORE COUNTER which is then reset to zero (by LOAD ZERO); it then counts until it becomes full (LS=1). The holding state is then replaced by the normal state, the contents of the SECTOR REGISTER are transferred back to the SECTOR COUNTER and everything goes back to normal again. The speed mode commands are then tested: If the cell is still not yet supposed to run at full speed, the normal state would last only one bit-time and the holding state would be again entered.

3. The single-step mode can only be entered manually by setting a switch properly. An R-S flip flop is used to debounce the pulser's push button. Another D-flip flop is essential as a conventional service flag to synchronize with the cell (since

the activation of the pulser is random).

In the single step mode, the cell is typically in the holding state. After every 256 bit-times, the speed mode commands are tested to see if the single step mode must remain; meanwhile the service flag (of the pulser) is also tested to see if a step is required. If the flag is down, nothing is done; otherwise the cell temporarily leaves the holding state (for just one bit-time) in a similar way as for the slow mode; the only additional thing to do is that the flag is reset.

4. The cell can be halted (if the automatic stop switch is set properly) electronically either by REF (a signal produced when the READ HEAD reaches a spot defined by an array of thumbwheel switches) or by HALT (an external signal coming from some test equipment).

The halt state is similar to the single step mode except that no step is generated. After every 256 bit-times, the cell tests to see if it has to remain in this state.

8.1.2.15 Logic Diagram L-16

This diagram shows how the READ HEAD and WRITE HEAD are applied to the CCD memories. Note that if DUAL HEAL = 0 (INSERT = 1), both HEADs are identical. The CCD chips are arranged in an X-Y matrix to reduce the number of drivers and wiring. The value of "4" of the number of columns is due to some physical restrictions.

8.1.2.16 Logic Diagram L-17

This diagram shows the timing for the beginning and end of the track.

1. A multiplexer permits the choice of many different memory sizes by the setting of a switch. A large AND gate is used to determine when the last minor loop is scanned (either by the READ HEAD or WRITE HEAD). A HEAD falls off the track when it exits from the last minor loop; this circumstance is indicated by RHOFF/WHOFF.

2. BLOCK_START is produced at the 1st sector of a minor loop when GAPE=1; this implies that no gap is allowed on the 1st sector. Consequently, the value of Block Size must be carefully calculated so that no gap can exceed 255 bits.

8.1.2.17 Logic Diagram L-18

This diagram shows clearly the signals other than CE and CS that drive the CCD chips.

1. Data to be written is sampled by Φ. CE1 guarantees that no data can overlap with the next bit and no data glitch can latch.

2. The Read Only mode can be entered by an appropriate setting of the Data Control Switch. In this mode, writing is prohibited and the contents of the track are preserved. It is of great importance for testing purposes.

3. The Manual Writing mode can also be entered by an appropriate setting of the

Data Control Switch. In this mode, data to be written on the track is defined by the Data Switch. It can be used with the Single Step mode and the Automatic Stop (by REF) mechanism to modify any bit anywhere on the track.

If N is the current reading of the 7-segment LED display, N indicates the current position of the READ HEAD; that of the WRITE HEAD is at N-X where X is either zero or the block size (in bits) depending on the value of DUAL HEAD (i.e. on the current C_op values). An attempt to write manually now will affect location 1+N-X.

8.1.2.18 Logic Diagram L-19

This diagram shows how the current position of the READ HEAD is displayed by 7-segment LEDS, and how it is compared with the setting of the thumbwheel switches to produce the signal REF which can be used to halt the cell or to trigger some test equipment.

8.1.3. LIST OF SIGNALS USED IN LOGIC DIAGRAMS L-1 TO L19

NOTE: The digits in parenthesis following the name of each signal refer to the L-number. The source of the signal is underlined.

A (3,17,13) Indicates that the mark-bit A of the current tuple has a mark.

a (14) The least significant bit (first bit) of the BASIC TIMING COUNTER. It is a square wave of half the frequency of MC.

a1 (9) Loaded by the CONTROLLER to indicate that the mark-bit A of all satisfied tuples must be marked or unmarked depending on the value of MARK and UNMARK.

a2, a3 (7) Loaded by the CONTROLLER to indicate that the presence or absence respectively of a mark on the mark-bit A is part of the Query List.

AD 1 ... AD 7 (1) Inverted from the RAP CONTROL BUS.

ADD (11,12) Decoded from C_op values; indicating that the cell must perform an addition on the track for all satisfied tuples.

ADR (7) Indicates that the cell-address is part of the Query List; this Feature is not necessary in RAP II.

ALIVE (2) A state of the cell indicating that it contains a well defined relation. If the cell is "destroyed", it becomes low and prevents the cell from being opened by any relation name.

AUTOFIN (2,8) A narrow pulse indicating that the cell discovers that it has completed its job.

B (3,7,13) Similar to A; but concerning the mark-bit B.

b(14) The second bit of the BASIC TIMING COUNTER (Cf. a).

b1 (9) Similar to a1; but concerning the mark-bit B.

b2, b3 (7) Similar to a2, a3; but concerning the mark-bit B.

BLOCKED (1,2,13) An important state of a cell. In this state, a cell is not sensitive to most commands of the CONTROLLER.

BLOCK_START (3,5,17) A narrow pulse preceding the first clock pulse of the first bit of every tuple while the cell is scanning its track.

BS(3) Simply a stretched form of BLOCK_START.

BT(10) A narrow signal used to write onto the I/O BUFFER and to transfer its contents to a parallel-to-serial converter.

C(3,17,13) Similar to A; but concerning the mark-bit C.

c(14) The third bit of the BASIC TIMING COUNTER.

c1 (9) Similar to a1; but concerning the mark-bit C.

c2, c3 (7) Similar to a2,a3; but concerning the mark-bit C.

CE1 (14,16,17,18) A periodic signal of the same frequency as Φ. It goes up twice in each cycle to enable reading and writing.

CHOICE (14) A square wave, half the frequency of OSC.

CK 0 ... CK 7 (1,2) Decoded from the RAP CONTROL BUS. They are effective for all cells of the system independently of their states.

CKa, CKb (11,12) Same as Φ; but restricted to one or two domains. Only used in Arithmetic Instructions.

C_op1 ... C_op4 (5,8,11,12) Loaded by the CONTROLLER; used to encode the job that the cell must do. A value of C_op4 = 1 indicates that an Arithmetic Instruction is involved.

CREATE (4,8,9,16) Decoded from C_op values to indicate that the CONTROLLER wants to define a new file (i.e. Relation).

CSX0 ... CSX3, CSY0 ... CSY15 (16,17) Decoded from the 6 most significant bits of the "current" HEAD (which can be READ or WRITE). Employed to select CCD chips.

D (3,7,13) Similar to A; but concerning the mark-bit D.

d (9) The third bit of the BASIC TIMING COUNTER (Cf. a).

d1 (9) Similar to a1; but concerning the mark-bit D.

d2, d3 (7) Similar to a2, a3; but conerning the mark-bit D.

DATA_IN (3,4,6,10,11,12,18) "Current" data read by the READ HEAD from the track.

DELETE (8,9) Decoded from C_op values; indicates that all satisfied tuples must be deleted (i.e. marked at the mark-bit D).

DF (3,7,8,10,13) Similar to A; but concerning the mark-bit DF.

DISJ (7,9) Loaded by the CONTROLLER to indicate that the Query List is in the disjunctive form; i.e. at least one item of the list must be satisfactory.

DL (3,4,5,6,8,10) A narrow pulse that follows the last clock pulse of every tuple.

DT 0 ... DT 15 (1,2,4,5,6,7,8,9,10,11,16) Inverted from the RAP DATA BUS.

DUAL_HEAD (16) A control signal to indicate that the READ HEAD and the WRITE HEAD are not identical. The WRITE HEAD must be calculated from the other HEAD by subtracting the contents of the BLOCK SIZE REGISTER. This signal must be connected to the complement of INSERT.

e (<u>14</u>) The most significant bit of the BASIC TIMING COUNTER.

E10, E18, E34 (3,<u>4</u>,6,11,12) Indicate that the cell is scanning an 8-bit, 16-bit or 32-bit long domain. Employed to rotate some shift registers and to control the WORD LENGTH COUNTER.

EOR (3,<u>4</u>,6,10,11,12) Indicates that the cell is scanning some domain. This is perhaps the most important signal of a cell. Its presence and stability indicate that the cell is essentially functioning.

EQUALITY (<u>5</u>,10,11) Covers a domain which is either an operand in an Arithmetic Instruction or retrieved in a partial retrieval mode.

EQUALITY2 (<u>5</u>,11) Covers the 2nd domain (if necessary) needed in an Arithmetic Instructions.

EX_DATA (<u>18</u>) Data sent from the cell to be written on the track. This signal must be connected to TTDATA.

EXFIN (<u>2</u>,5,6,7,8,9) A narrow pulse generated by the CONTROLLER to terminate the activities of the cell and to erase many of its registers.

EXRUN (<u>2</u>,4,8,10,11,13) A narrow pulse controlled by the CONTROLLER to initiate the cell to start scanning its track to do its job (after all necessary parameters have been supplied).

EX_SLOW (<u>15</u>) A control signal for testing purpose only. When being high, it causes the speed of the CCD Track to slow down by 257 times. It has no effect if the speed is set at Single Step Mode.

f1, f2, f3 (<u>6</u>) Values loaded by the CONTROLLER to indicate that a domain must be smaller than some constant (which is also loaded by the CONTROLLER) as part of the Query List. The digital suffix refers to the chosen Domain Comparator Unit.

FAST (<u>15</u>) Decoded from the SPEED CONTROL SWITCH to indicate that the cell must run at its highest speed.

g1, g2, g3 (<u>6</u>) Similar to f1, f2, f3; but equality is required.

GAPE (<u>3</u>,4,5,6,11,17) A signal that goes low from the beginning of every tuple until its end when it will go high again. This remains true even if two consecutive tuples have no space between them.

h1, h2, h3 (<u>6</u>) Similar to f1, f2, f3; but superlative comparison is required.

HALT (<u>15</u>) A control signal for testing purpose. It can bring the scanning of the CCD Track to a halt if the AUTOMATIC STOP CONTROL is set properly.

HMA (<u>7</u>) Indicates that the cell-address has been compared successfully with some value supplied by the CONTROLLER as part of the Query List. This feature is not needed in RAP II.

INS 0 ... INS 7 (<u>1</u>,2) Decoded from the RAP CONTROL BUS. They are effective for all open cells.

INS_DATA (9,<u>10</u>) Data to be written on the track when the cell is processing an INSERT Instruction. For each domain, they come as a (typically broken) stream of pulses; absence of a pulse indicates a zero.

INSERT (<u>8</u>,9,10,16) Decoded from C_op values to indicate that a specified number of vacant tuples on the Track must be filled with the contents of the I/O BUFFER.

INT-1, INT-2 (5,<u>8</u>,10,12) Values loaded by the CONTROLLER if the relevant task is a 2-operand instruction (i.e. ADD, SUBTRACT or REPLACE)

KEY_ENABLE (<u>1</u>,2,4,5,6,10,13) A pulse generated by the CONTROLLER. Other lines of RAP CONTROL BUS and RAP DATA BUS must remain constant while KEY_ENABLE is activated.

LC1, LC2, (<u>3</u>,4,5,6) Two consecutive bit-time signals that precede and follow all domains.

LCE (<u>17</u>) A signal indicating that the last CCD chip is being used. It helps to detect when the R/W HEADS go beyond the physical end of the Track.

LOAD I/D BUFFER (5,<u>8</u>,10) Decoded from the C_op values to indicate that the cell must look for not more than a specified number of satisfied tuples and store them (or up to three domains per tuple) in the I/O BUFFER.

LOAD_ZERO (<u>15</u>) A narrow pulse to load the SECTOR COUNTER with zero (to start a count of 256 bit-times during which the Track behaves as if it does not move).

LOCK_OPEN (1,<u>2</u>) An important state of a cell: Not in this state, it cannot recieve most commands of the CONTROLLER.

LS (<u>15</u>,17) Decoded from the SECTOR COUNTER to indicate that it is full.

MANUAL (14,<u>18</u>) Decoded from the DATA CONTROL SWITCH indicating that data to be written on the Track must come from the DATA SWITCH.

MARK (<u>9</u>) A value loaded by the CONTROLLER indicating that all satisfied tuples must be marked according to a1, b1, c1 and d1.

MAX, MIN (11,<u>12</u>) Decoded from C_op values; indicating that the cell must find the max or min value of the specified domain of all satisfied tuples and put it in the RESULT REGISTER.

MC (<u>14</u>) Master clock; a 20 Mhz square wave. Every activity is synchronized with MC.

MOR (<u>3</u>,4,5) A narrow pulse occuring immediately after the last clock pulse of every domain.

OR1P (9,<u>12</u>) Similar to INS_DATA; but concerning either ADD, SUB or REP.

OSC (<u>14</u>,15) A narrow pulse marking the beginning of every bit-time.

Φ (3,4,6,10,11,12,<u>14</u>,17,18) The basic clock pulse of the cell.

Φ1, Φ2, Φ3, Φ4 (<u>14</u>,18) Four phrases required to shift the CCD chips.

ø1, ..., ø7 (<u>3</u>,4) The first seven bit-times of every tuple.

øc1, øc2, øc3 (<u>6</u>) Same as Φ; but restricted to some selected domain; used to rotate the comparand in the Domain Comparator Units.

PLC1, PLC2 (<u>3</u>,6) The first and second bit-time after the last bit of every domain.

PLD 0, ..., PLD 15 (<u>1</u>,2,4,6,7,8,9,10,11,16) Decoded from the RAP CONTROL BUS. They are effective only to open cells that are neither blocked, nor rejected, nor running.

POLL_IN, POLL_OUT (<u>2</u>) Indicate the input and output points of the daisy-chained priority line that runs through all cells in the System. This line is needed only in large systems to replace a polling-by-address scheme.

Q1 (<u>6</u>,7) Indicates that the Domain Comparator Unit #1 is chosen and the result of the comparison (as specified by f1, g1, h1) is positive.

Q2, Q3 (<u>6</u>,7) Similar to Q1; but applied to the other remaining units.

QDCJ (<u>7</u>,8,10) Indicates that the "current" tuple contains valid data and it satisfies the Query List. In the case of data retrieval, a low value of QDCJ could indicate that the limit of tuples to be retrieved has been reached.

QMRK (<u>7</u>) Indicates that the Query List asks for the presence of at least one mark (as specified by a2, b2, c2, d2) and that the current tuple is satisfactory as far as the presence of marks is concerned.

QUMRK (<u>7</u>) Similar to QMRK; but related to the absence of marks.

R_ADDRESS, R_ALU, R_BC,R_I/O, R_RN, R_SC, R_STATUS (COND) (1,10,13) Decoded from the RAP CONTROL BUS. They are effective for open cells that are neither blocked nor rejected. Employed to put the Address, RESULT REGISTER, BUFFER COUNTER, I/O BUFFER, RELATION NAME REGISTER, S_COUNTER and STATUS respectively on the RAP DATA BUS.

R_STATUS (UNC) (1,13) Decoded from the RAP CONTROL BUS. It is effective for all cells independently of their states. Employed to put the STATUS on the RAP DATA BUS.

RAP CONTROL BUS (1) A set of 8 lines of control flowing from the CONTROLLER to all cells.

RAP DATA BUS (1,13) A set of 16 bilateral lines to transfer data from the CONTROLLER to all cells and back. Not used to transfer data between cells.

RBIT (9) A narrow pulse indicating that the "current" mark-bit (relative to the WRITE HEAD) must be erased.

READ_ONLY (18) Decoded from the DATA CONTROL SWITCH to indicate that the contents of the CCD Track must not be altered. This is of great value for testing purpose.

REF (15,19) A signal indicating that the "current" location on the Track matches with the setting of the thumbwheel switches. This is also of great value for testing purpose.

REJECTED (1,2,13) An important state of a cell to indicate that it has been served.

RELOAD (15) A narrow pulse used to restore the contents of the SECTOR COUNTER after it has counted a multiple of 256 bit-times during which interval the CCD Track behaves as if it is standing still.

REN (9,14) Indicates that the "current" bit (relative to the WRITE HEAD) must be rewritten.

REP (11,12) Decoded from the C_op values to indicate that the specified domain of all satisfied tuples must be replaced either by a constant or by another domain.

RHOFF (8,7,17) Indicates that the READ HEAD has gone beyond the physical end of the CCD track.

RSQ (3,4,5,10,12,14) Another clock phase that occurs slightly later than Φ. Employed mainly to truncate some signals so that they do not overlap with the next bit-time.

RUNNING (1,2,4,8,10,11,13,17) Indicates that the cell is executing its job and therefore the CONTROLLER cannot do many things to the cell.

SAMPLE (14,16,18,19) A narrow pulse that occurs slightly before Φ. Used to sample data from the CCD Track.

R_ADDRESS, R_ALU, R_BC,R_I/O, R_RN, R_SC, R_STATUS (COND) (1,10,13) Decoded from the RAP CONTROL BUS. They are effective for open cells that are neither blocked nor rejected. Employed to put the Address, RESULT REGISTER, BUFFER COUNTER, I/O BUFFER, RELATION NAME REGISTER, S_COUNTER and STATUS respectively on the RAP DATA BUS.

R_STATUS (UNC) (1,13) Decoded from the RAP CONTROL BUS. It is effective for all cells independently of their states. Employed to put the STATUS on the RAP DATA BUS.

RAP CONTROL BUS (1) A set of 8 lines of control flowing from the CONTROLLER to all cells.

RAP DATA BUS (1,13) A set of 16 bilateral lines to transfer data from the CONTROLLER to all cells and back. Not used to transfer data between cells.

RBIT (9) A narrow pulse indicating that the "current" mark-bit (relative to the WRITE HEAD) must be erased.

READ_ONLY (18) Decoded from the DATA CONTROL SWITCH to indicate that the contents of the CCD Track must not be altered. This is of great value for testing purpose.

REF (15,19) A signal indicating that the "current" location on the Track matches with the setting of the thumbwheel switches. This is also of great value for testing purpose.

REJECTED (1,2,13) An important state of a cell to indicate that it has been served.

RELOAD (15) A narrow pulse used to restore the contents of the SECTOR COUNTER after it has counted a multiple of 256 bit-times during which interval the CCD Track behaves as if it is standing still.

REN (9,14) Indicates that the "current" bit (relative to the WRITE HEAD) must be rewritten.

REP (11,12) Decoded from the C_op values to indicate that the specified domain of all satisfied tuples must be replaced either by a constant or by another domain.

RHOFF (8,7,17) Indicates that the READ HEAD has gone beyond the physical end of the CCD track.

RSQ (3,4,5,10,12,14) Another clock phase that occurs slightly later than Φ. Employed mainly to truncate some signals so that they do not overlap with the next bit-time.

RUNNING (1,2,4,8,10,11,13,17) Indicates that the cell is executing its job and therefore the CONTROLLER cannot do many things to the cell.

SAMPLE (14,16,18,19) A narrow pulse that occurs slightly before Φ. Used to sample data from the CCD Track.

SATISFIED (8,9,10,11) Indicates that the "current" (relative to the WRITE HEAD). tuple is satisfied and must be perhaps marked, updated, retrieved, etc.

SAT_ONCE (11,12,13) Indicates that the cell has seen at least one satisfied tuple since it was initiated to run.

SBIT (9) A narrow pulse indicating that the "current" mark-bit (relative to the WRITE HEAD) must be marked.

SINGLE (15) Decoded from the SPEED CONTROL SWITCH to indicate that the speed is in the Single Step Mode; i.e. the PULSER BUTTON must be activated to advance the Track.

SL-1 (5,8,10) Loaded by the CONTROLLER to indicate that the data retrieval instruction is in the partial mode; i.e. only up to three domains from each tuple must be stored in the I/O BUFFER.

SL-4, SL-5, SL-6 (6,7) Loaded by the CONTROLLER to indicate that the Domain Comparator Unit #1, #2, #3 respectively is chosen to execute the Query List.

SLD 0, ..., SLD 7 (1,4,5,6,10) Decoded from the RAP CONTROL BUS to load serially some registers. They are effective for all open cells that are neither blocked nor rejected nor running.

SPACE_COUNT (8) Decoded from the C_op values to indicate that the cell must count to see how many empty spaces it has left on its track. The answer is left in the S_COUNTER.

SQ1, ... SQ7 (3,6,8,9,10,11) Narrow pulses (slightly wider than Φ) that mark the first seven bits of every tuple.

STOP (15) A signal occuring only in the Single Step Mode. It indicates that the cell is waiting for someone to activate the PULSER BUTTON.

STORE (14,15,17) A signal coverning the interval during which the CCD Track behaves as if it is not spinning. SAMPLE, Φ, RSQ, CE1, WE are not generated in this interval which must be a multiple of 256 "bit-times".

SUB (11,12) Decoded from the C_op values to indicate that the cell must perform a subtraction on the track for all satisfied tuples.

SUM (11,12) Decoded from the C_op values to indicate that the cell must calculate the sum of the specified domain of all satisfied tuples and leave it in the RESULT REGISTER.

TAKE NEW (11) Employed only in MAX/MIN to indicate that current (specified) domain has a better value that must be stored if the relevant tuple qualifies.

TKE (4,7,8,10) Indicates that either the track has no more valid data or that (in the case of data retrieval) the limit of retrieval has been reached.

TRACK_START (17) A narrow pulse to mark the moment the CCD Track gets synchronized with the cell when it is running (i.e. it occurs just before the clock pulse of the first bit of the Track).

TRIGGER (14,15,16,17) Decoded from the BASIC TIMING COUNTER. It comes slightly later than OSC. It changes the CCD location pointer from WRITE HEAD to READ HEAD.

TSYN (3,4,14,15,17) Indicates that the cell is scanning its track and is synchronized with it.

TTDATA (9,18) The data that must be written on the track if REN is high. See also EX_DATA.

UNMARK (9) Similar to MARK; but for unmarking purpose. If both MARK and UNMARK are high, UNMARK has no effect.

UPDATE (9,11,12) Covers the domain that must be updated of all satisfied tuples in the instructions ADD, SUBTRACT, REPLACE.

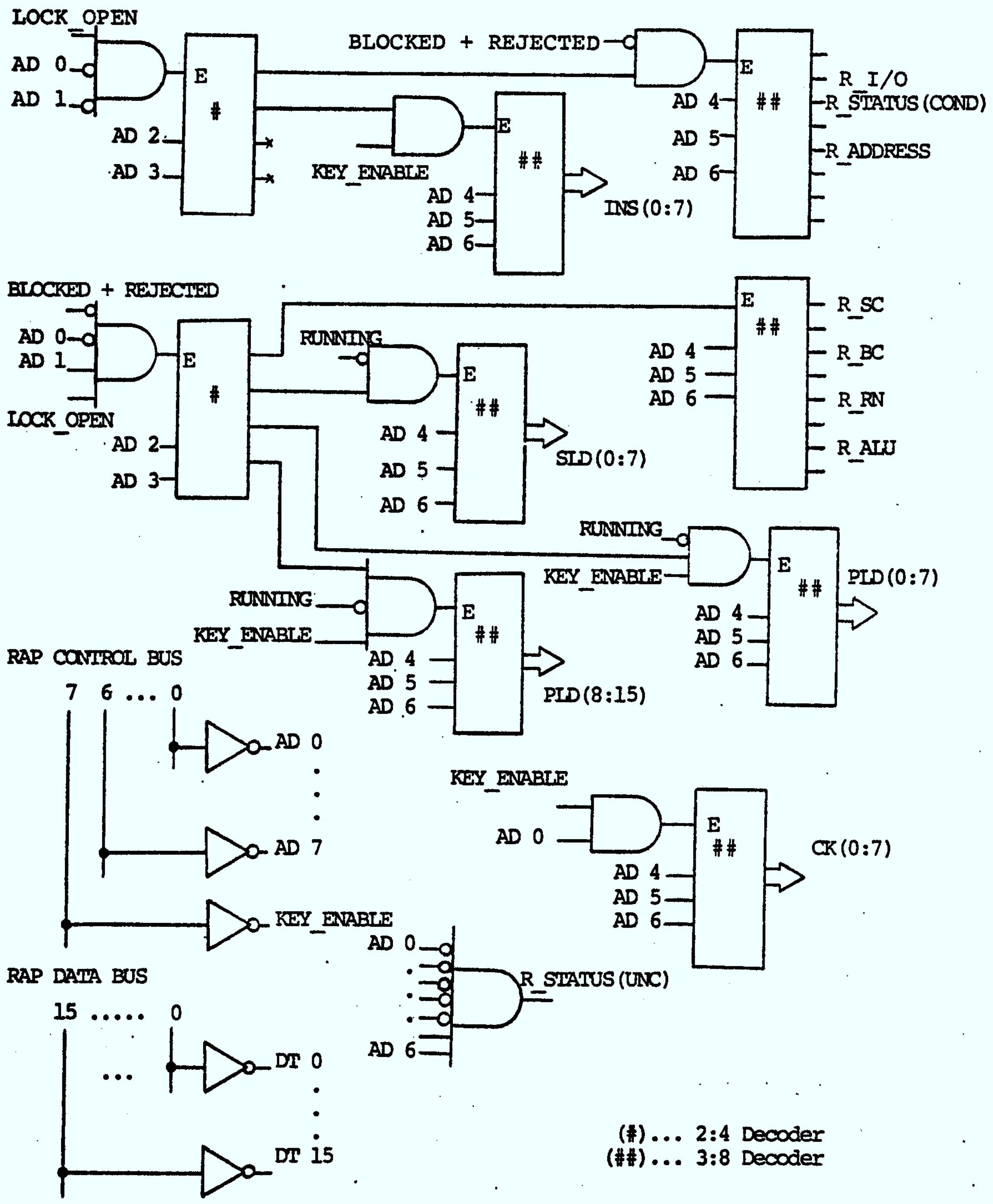
WE (14,18) The "enable" signal for writing on CCD chips.

WHOFF (8,17) A narrow pulse to indicate that the WRITE HEAD has gone beyond the physical end of the track and therefore producing the AUTOFIN signal.

YO, Y1, Y2 (5) Decoded from a 2-bit counter to connect one of 3 registers to a comparator to produce the EQUALITY and EQUALITY2 signals.

8.1.4 LOGIC DIAGRAMS L-1

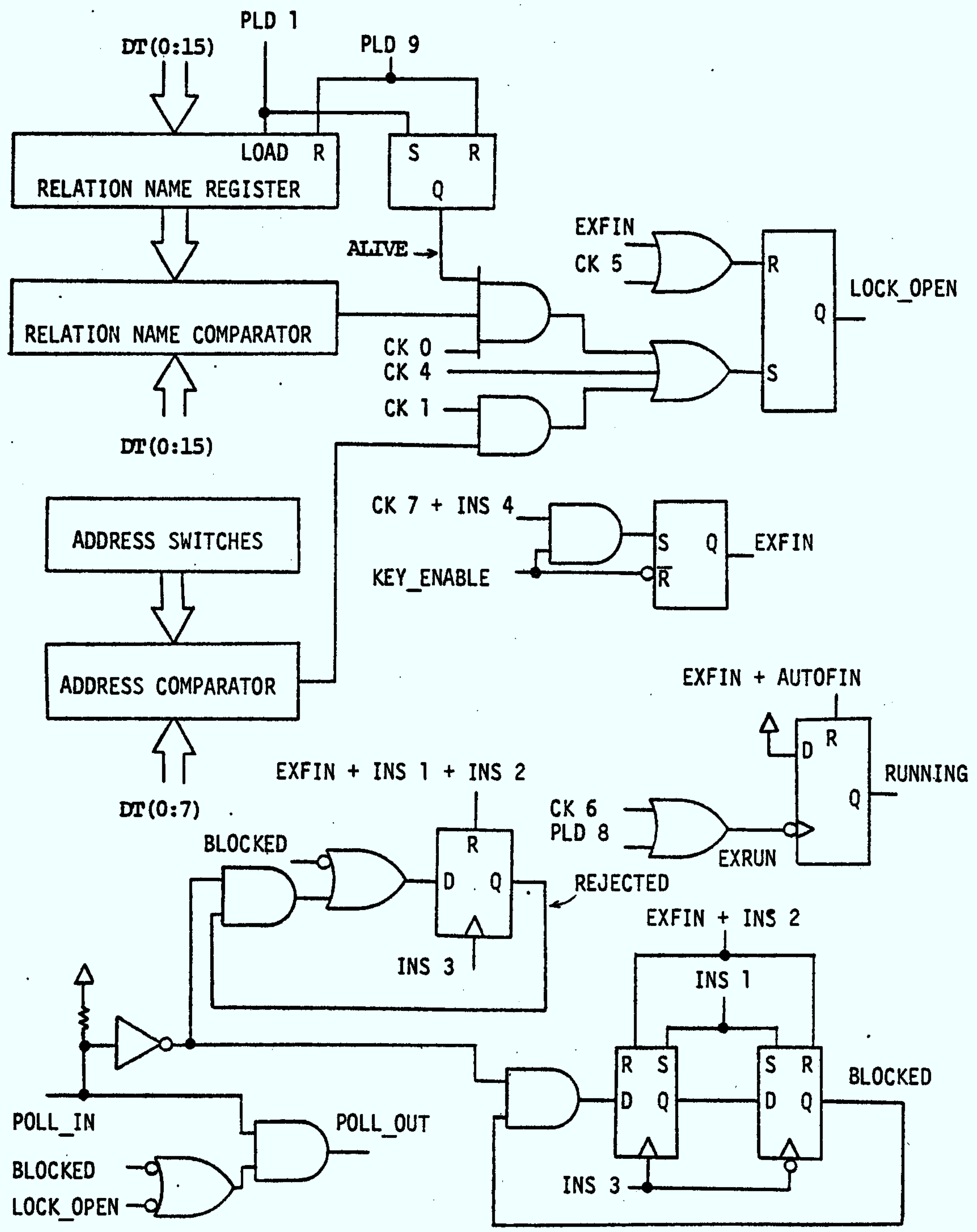
CELL INTERFACE: BUS RECEIVERS, CONTROL DECODERS



(#)... 2:4 Decoder
(##)... 3:8 Decoder

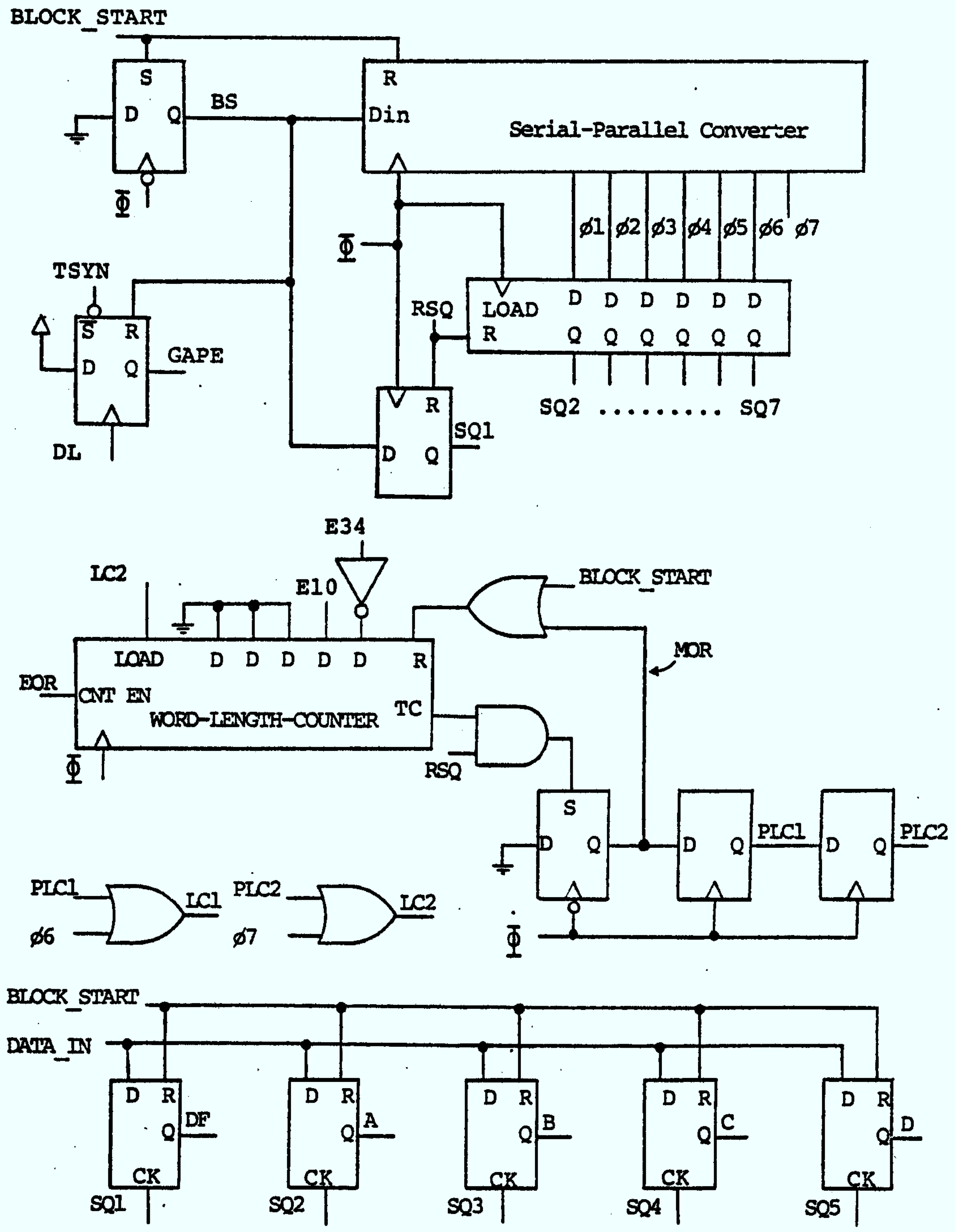
LOGIC DIAGRAM L-2

CELL INTERFACE: CONTROL OF IMPORTANT STATES



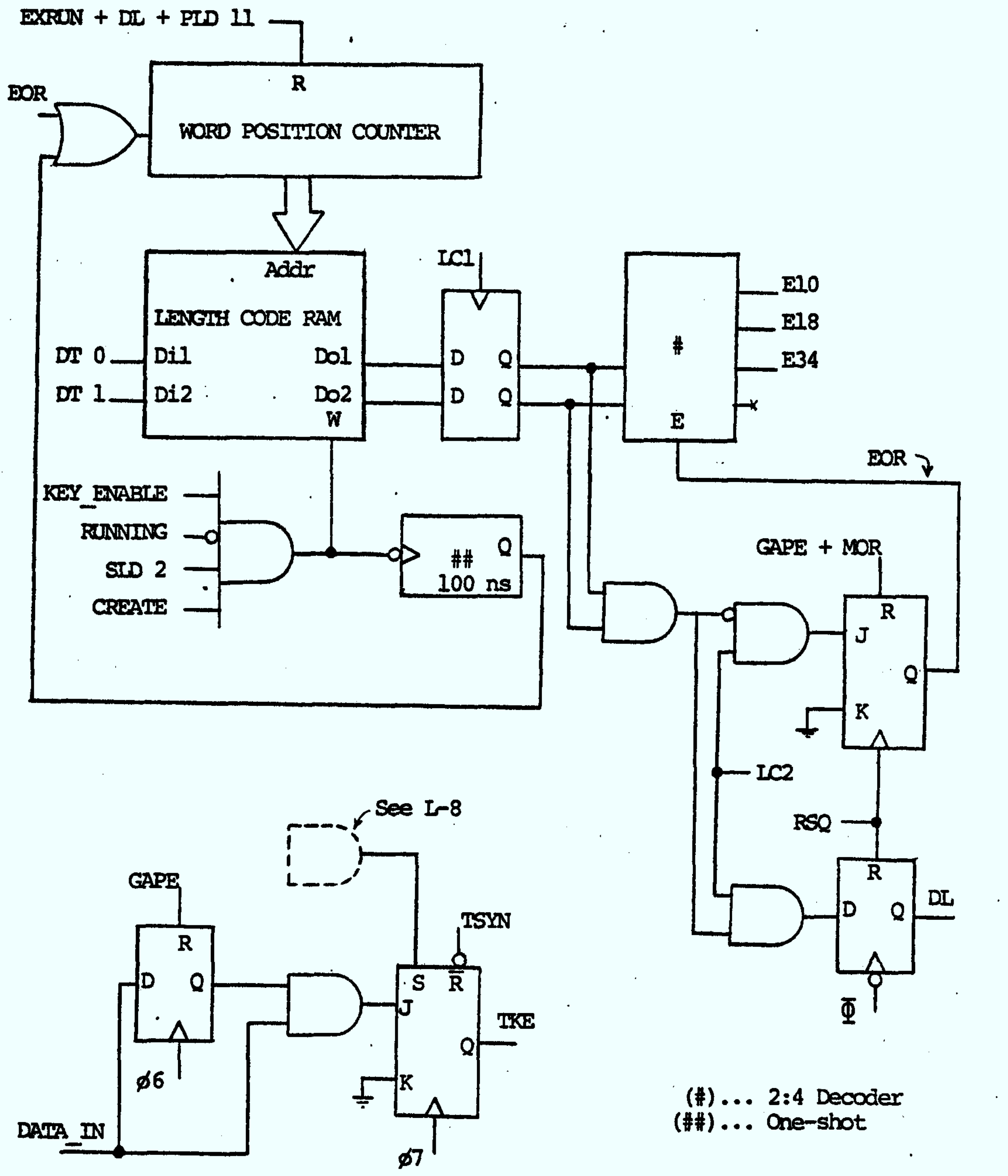
LOGIC DIAGRAM L-3

SYNCHRONIZER: MARK-BITS, LENGTH CODES, DOMAINS.



LOGIC DIAGRAM L-4

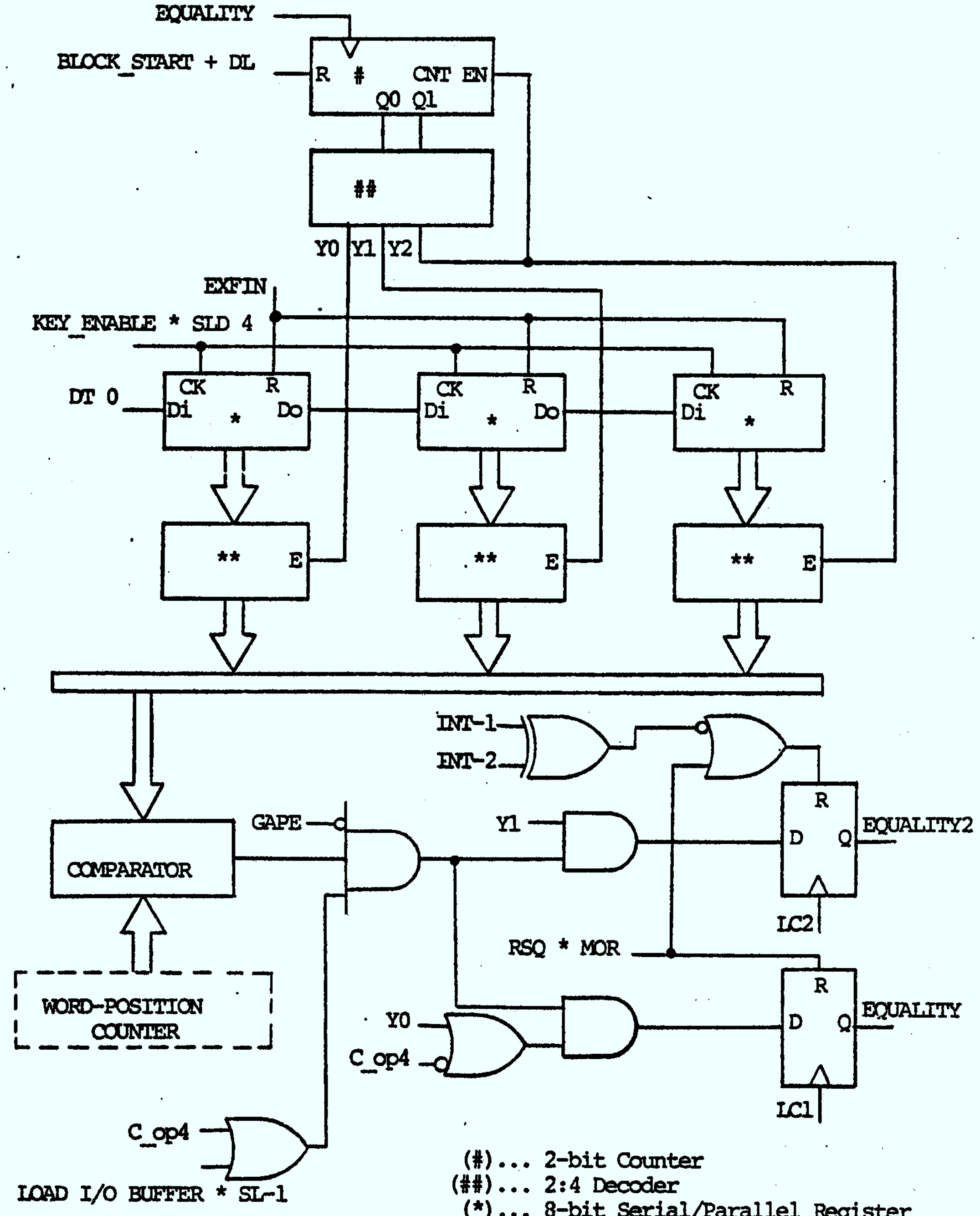
SYNCHRONIZER: LENGTH CODE RAM, DOMAIN INDICATORS, DELIMITER, TKE



(#)... 2:4 Decoder
(##)... One-shot

LOGIC DIAGRAM L-5

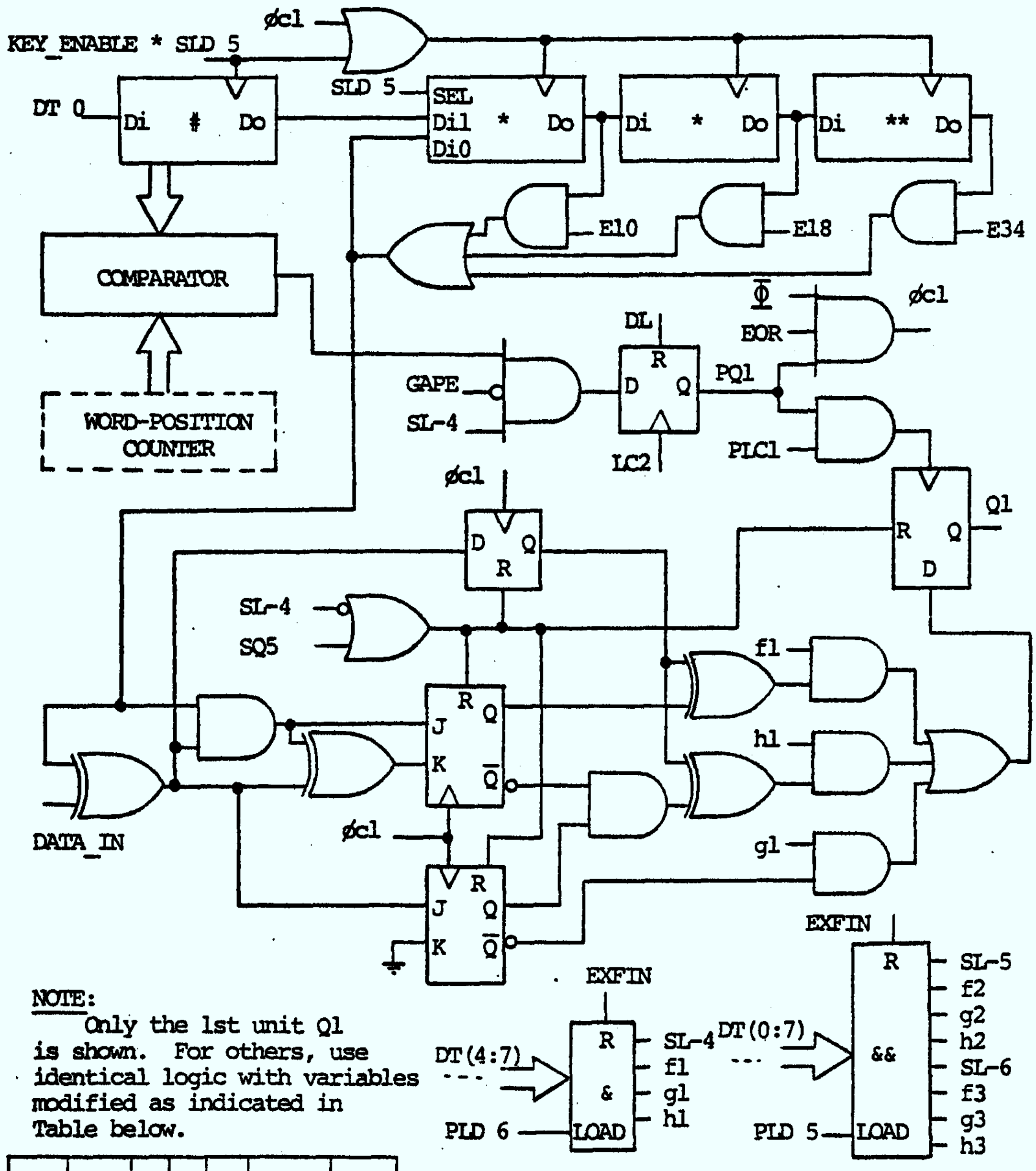
SYNCHRONIZER: DOMAIN SELECTOR



(#)... 2-bit Counter
(##)... 2:4 Decoder
(*)... 8-bit Serial/Parallel Register
(**)... Tri-state Buffers

LOGIC DIAGRAM L-6

QUERY ANALYZER: DOMAIN COMPARATOR UNITS



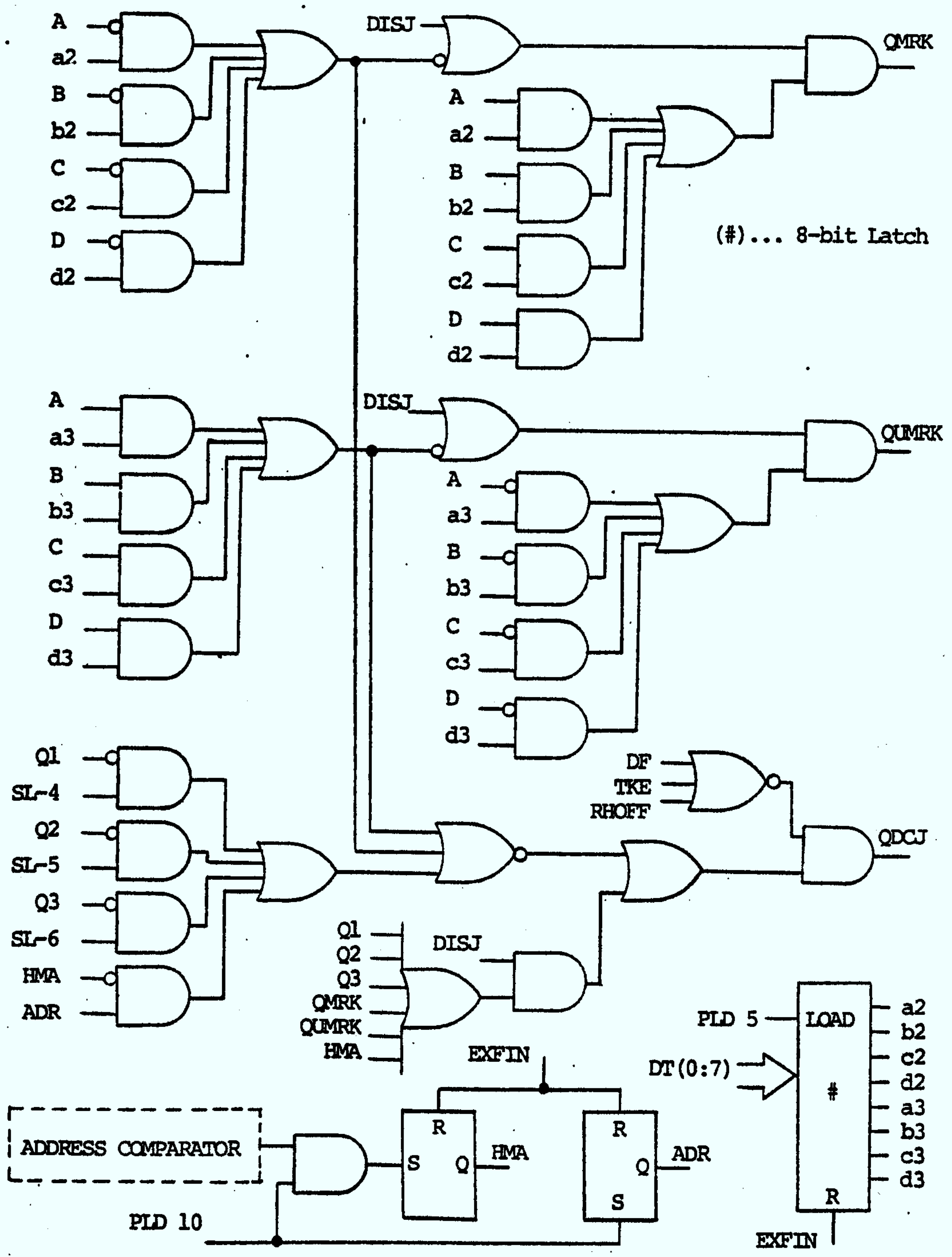
NOTE:
Only the 1st unit Q1 is shown. For others, use identical logic with variables modified as indicated in Table below.

Q1	SL-4	f1	g1	h1	SLD 5	øc1
Q2	SL-5	f2	g2	h2	SLD 6	øc2
Q3	SL-6	f3	g3	h3	SLD 7	øc3

(#)... 8-bit Serial/Parallel Register
(*)... 8-bit Shift Register
(**)... 16-bit Shift Register
(&)... 4-bit Latch
(&&)... 8-bit Latch

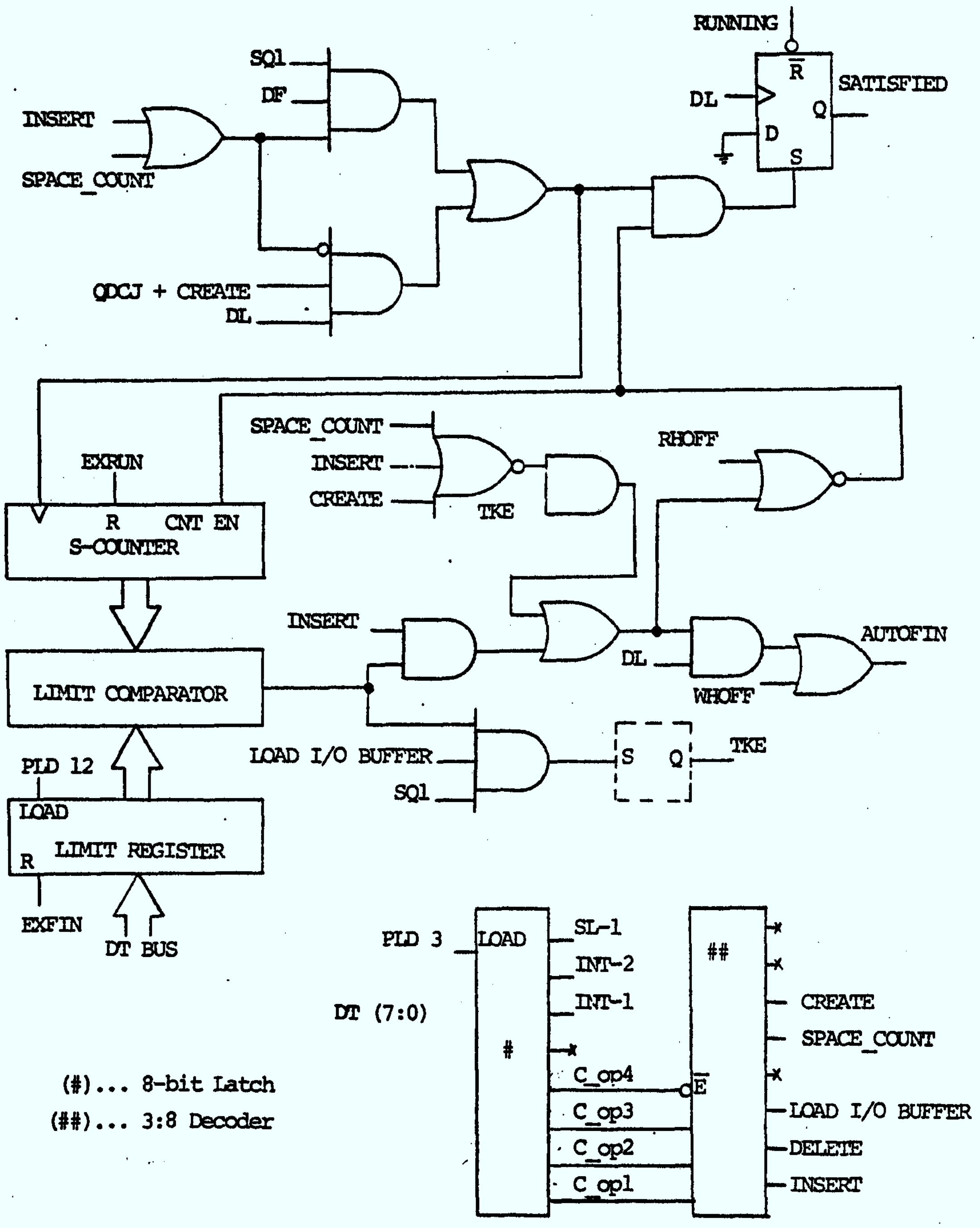
LOGIC DIAGRAM L-7

QUERY ANALYZER: TERMS EVALUATOR



LOGIC DIAGRAM L-8

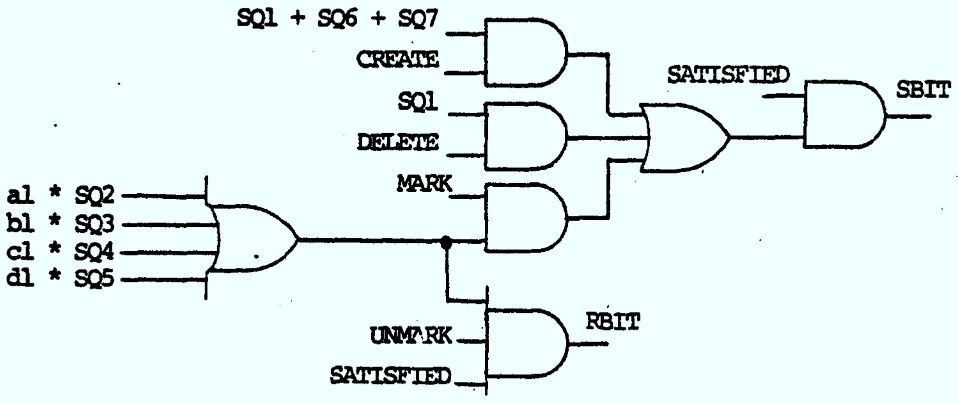
SYNCHRONIZER: GENERATION OF "SATISFIED" AND "AUTOFIN"

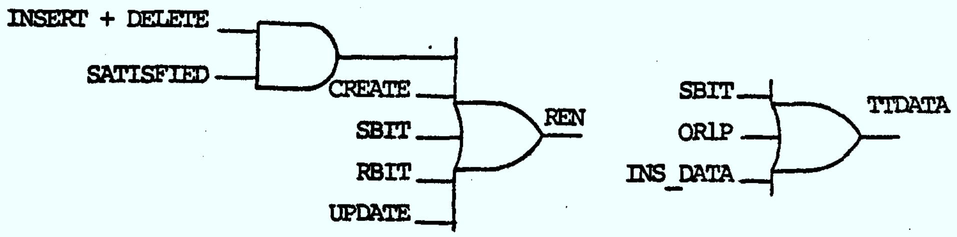


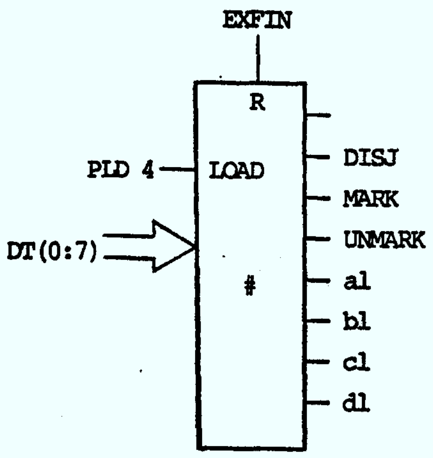
(#)... 8-bit Latch
(##)... 3:8 Decoder

LOGIC DIAGRAM L-9

UPDATE CONTROL UNIT



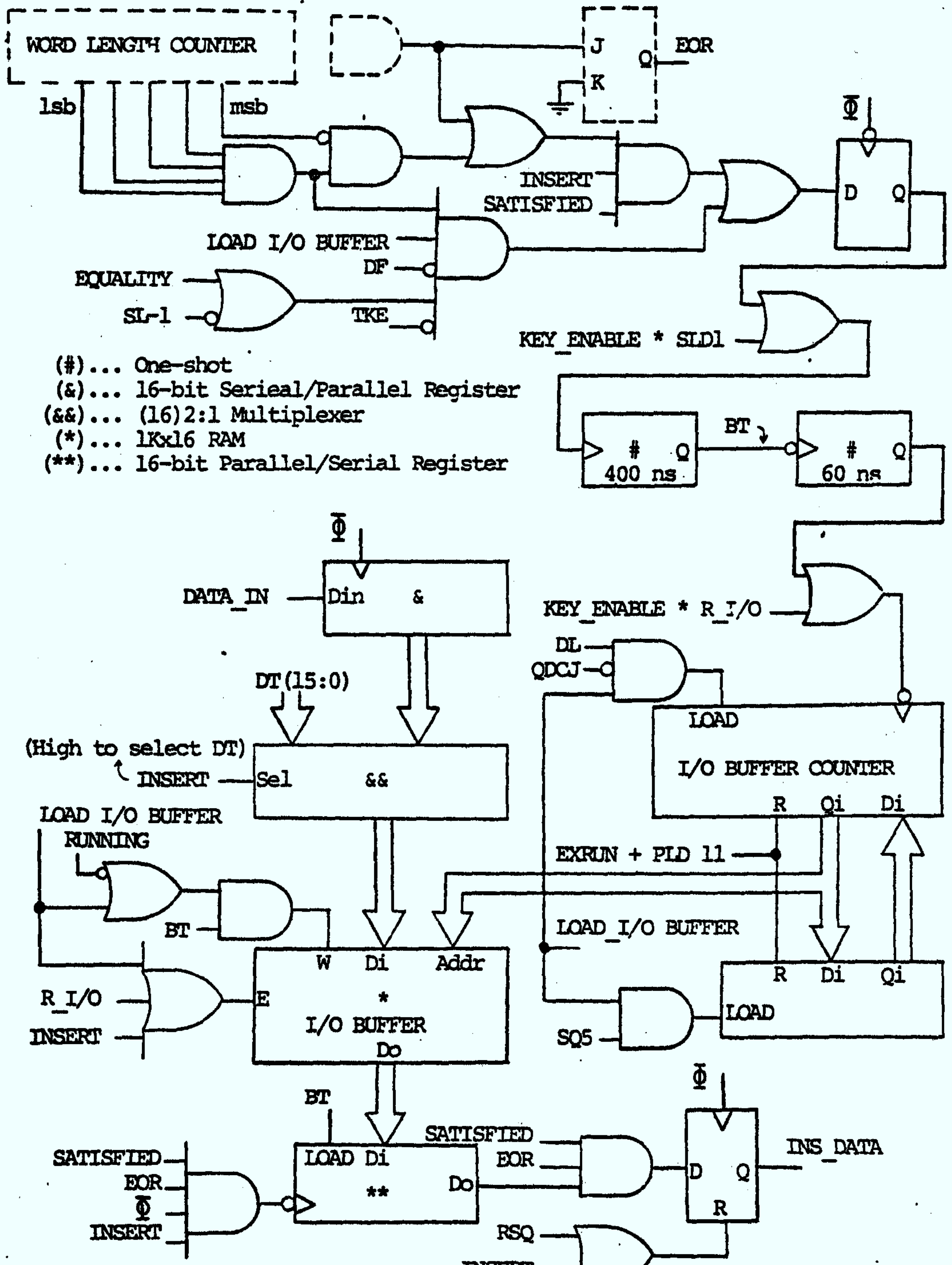




(#)... 8-bit Latch

LOGIC DIAGRAM L-10

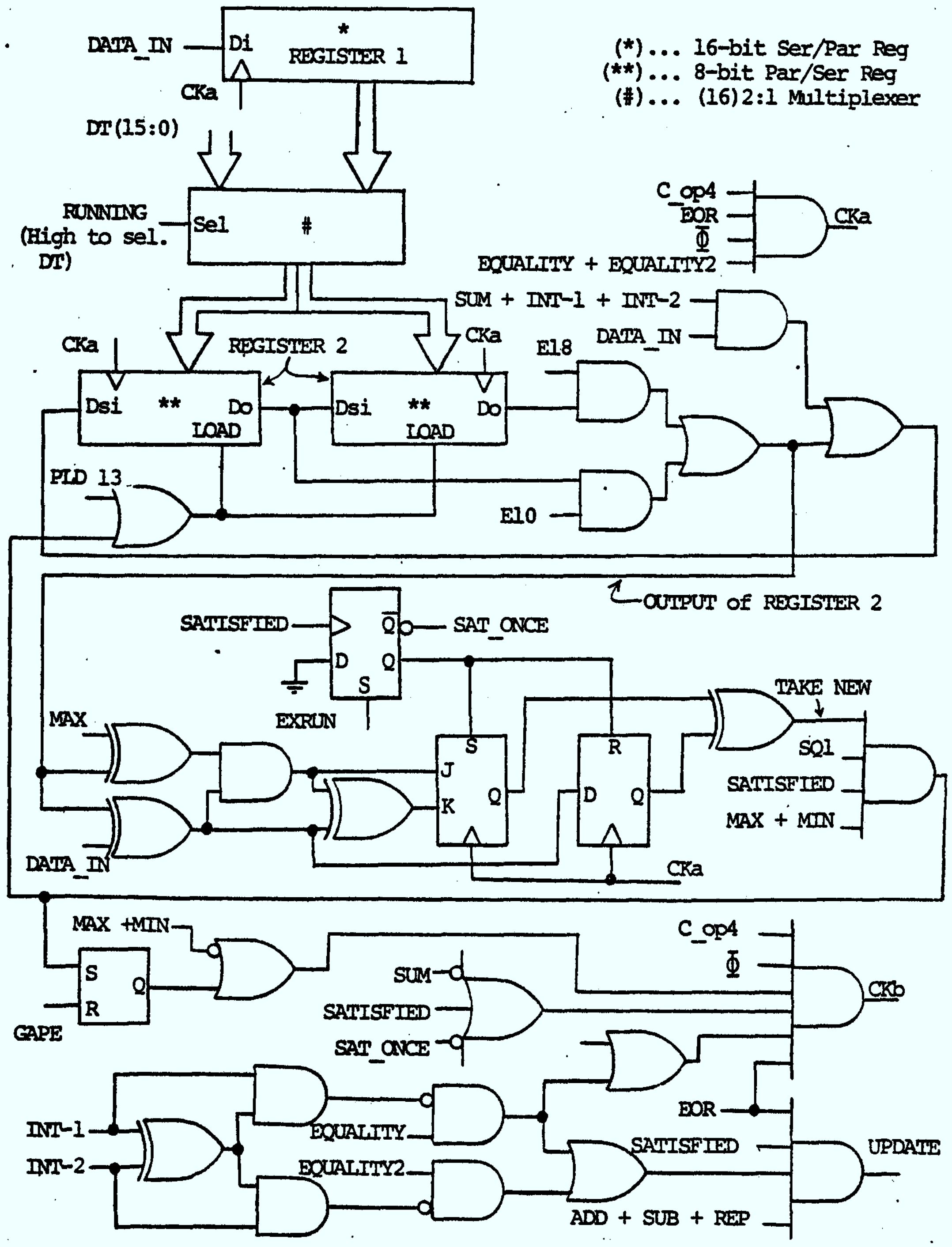
I/O BUFFER



(#)... One-shot
(&)... 16-bit Serieal/Parallel Register
(&&)... (16)2:1 Multiplexer
(*)... 1Kx16 RAM
(**)... 16-bit Parallel/Serial Register

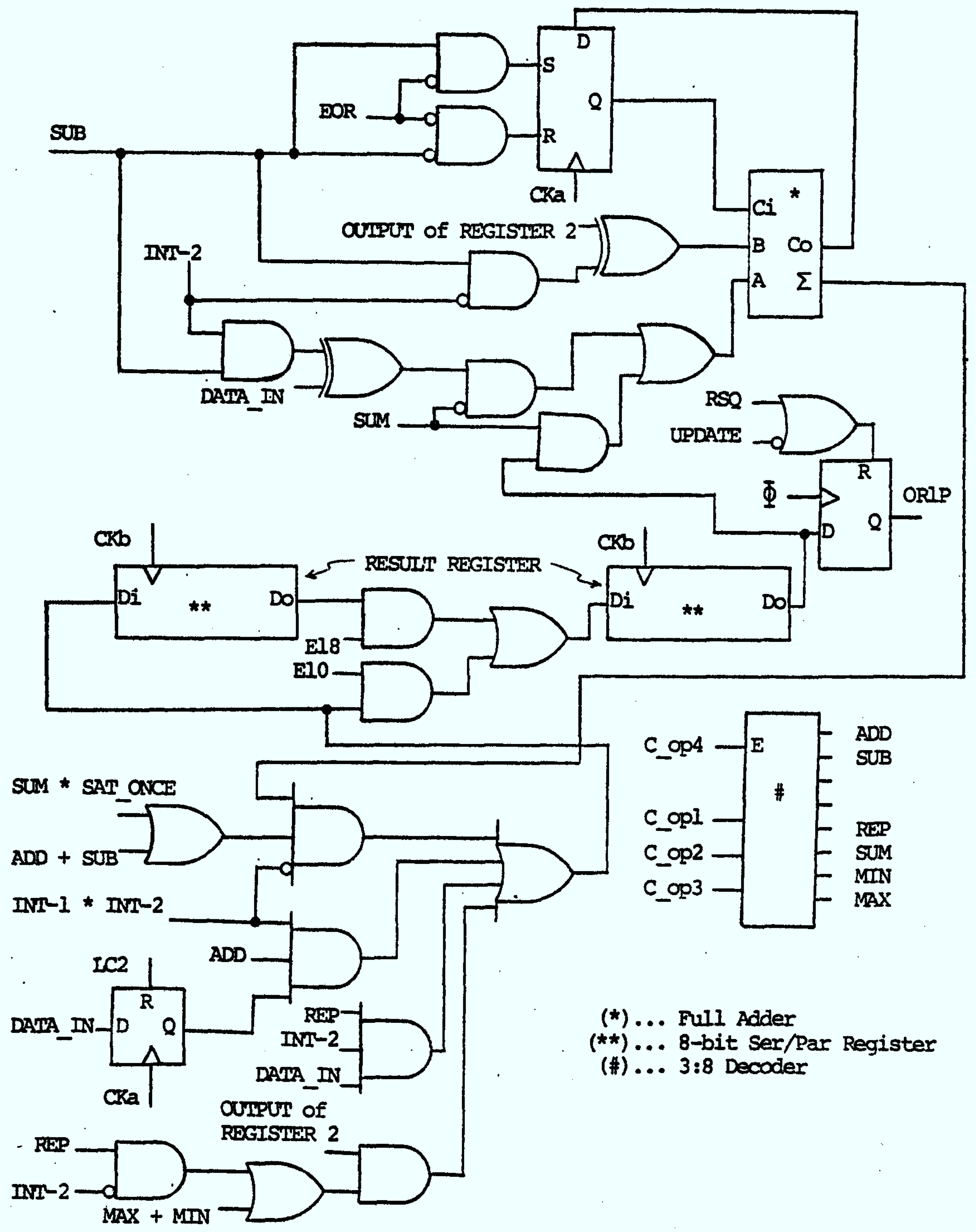
LOGIC DIAGRAM L-11

ARITHMETIC UNIT: REGISTER 1, REGISTER 2, COMPARATOR, SHIFT CONTROL.



LOGIC DIAGRAM L-12

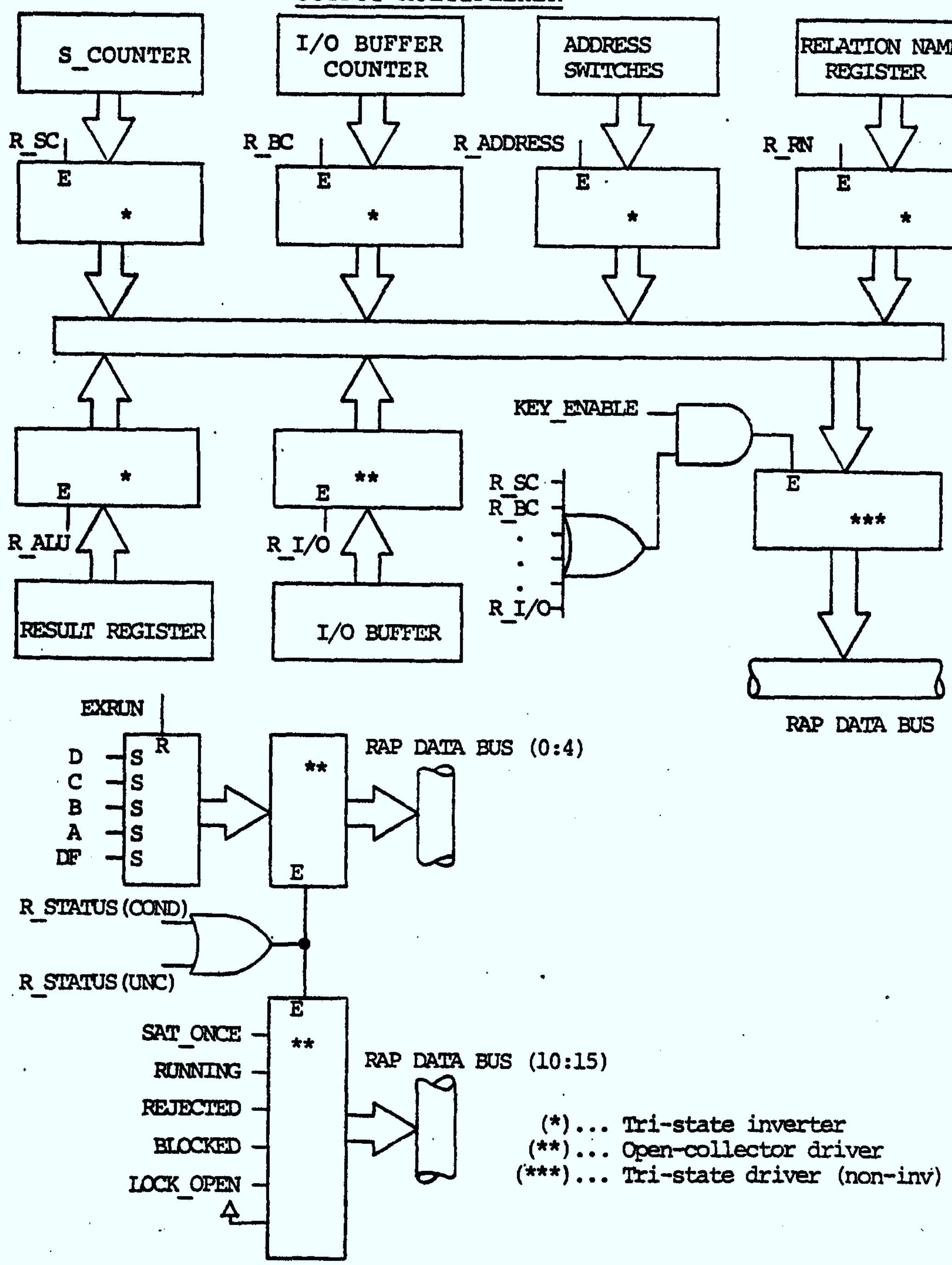
ARITHMETIC UNIT: RESULT REGISTER, ADDER



(*)... Full Adder
(**)... 8-bit Ser/Par Register
(#)... 3:8 Decoder

LOGIC DIAGRAM L-13

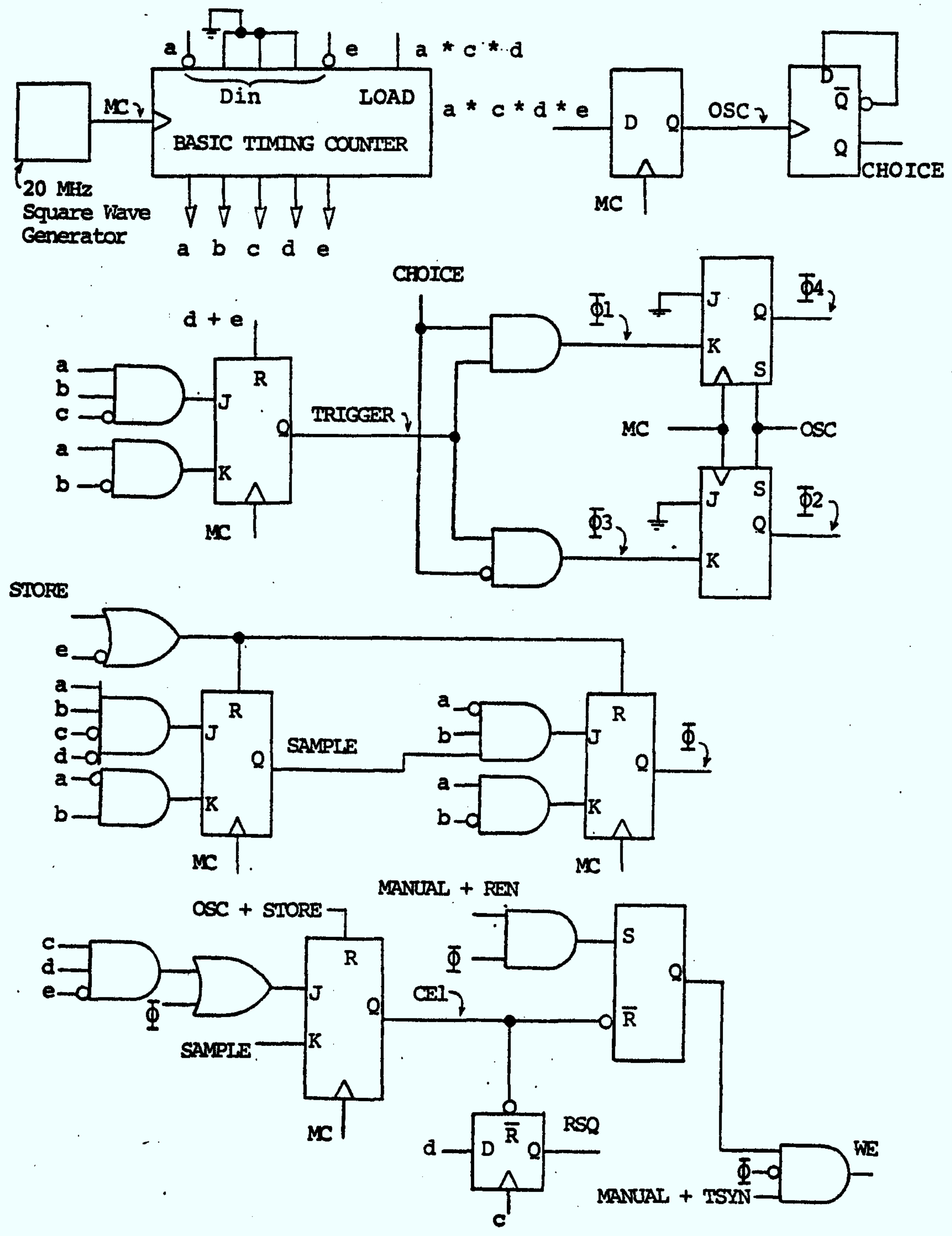
OUTPUT MULTIPLEXER



(*)... Tri-state inverter
(**)... Open-collector driver
(***)... Tri-state driver (non-inv)

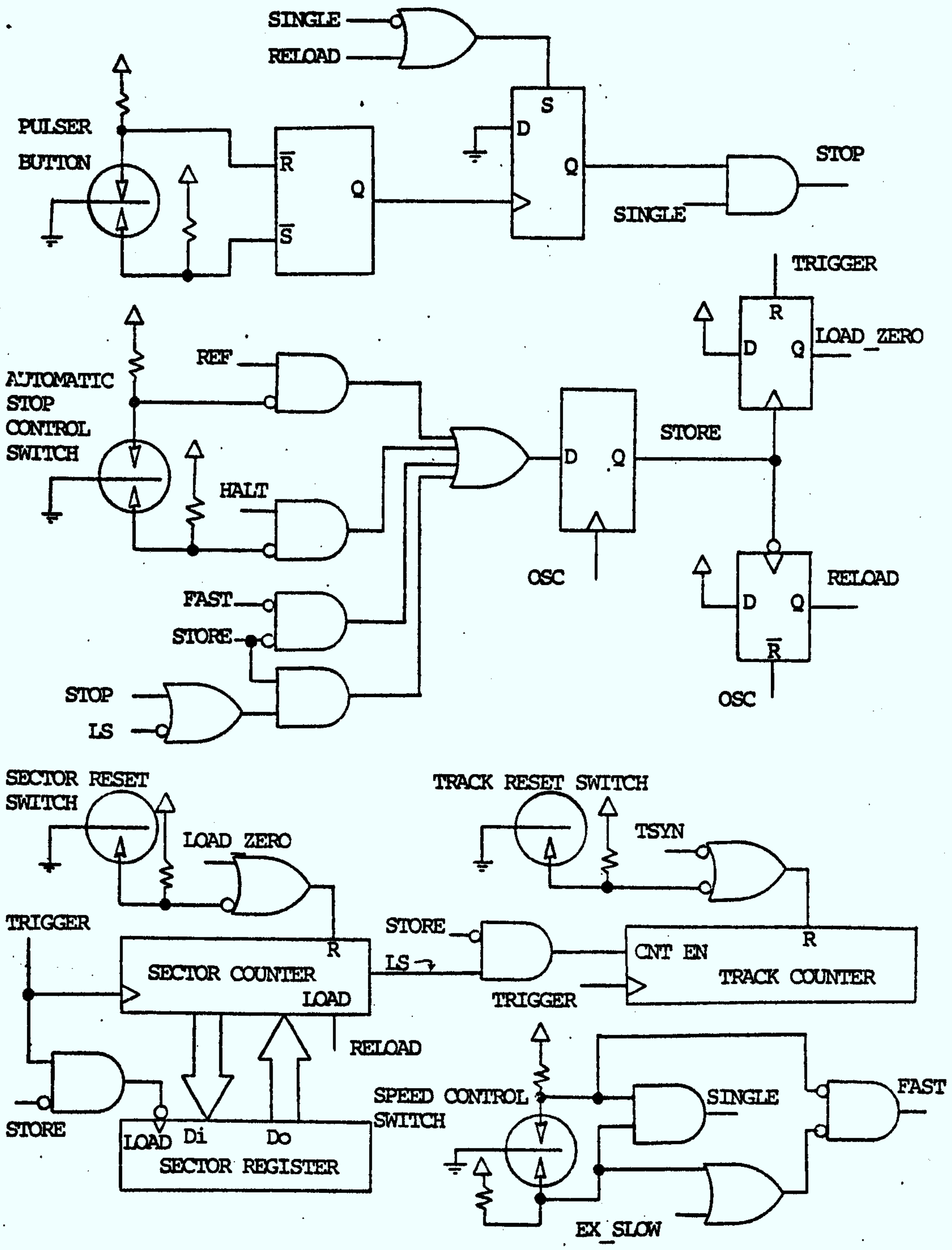
LOGIC DIAGRAM L-14

SYNCHRONIZER: PHASE GENERATORS FOR CCD MEMORIES



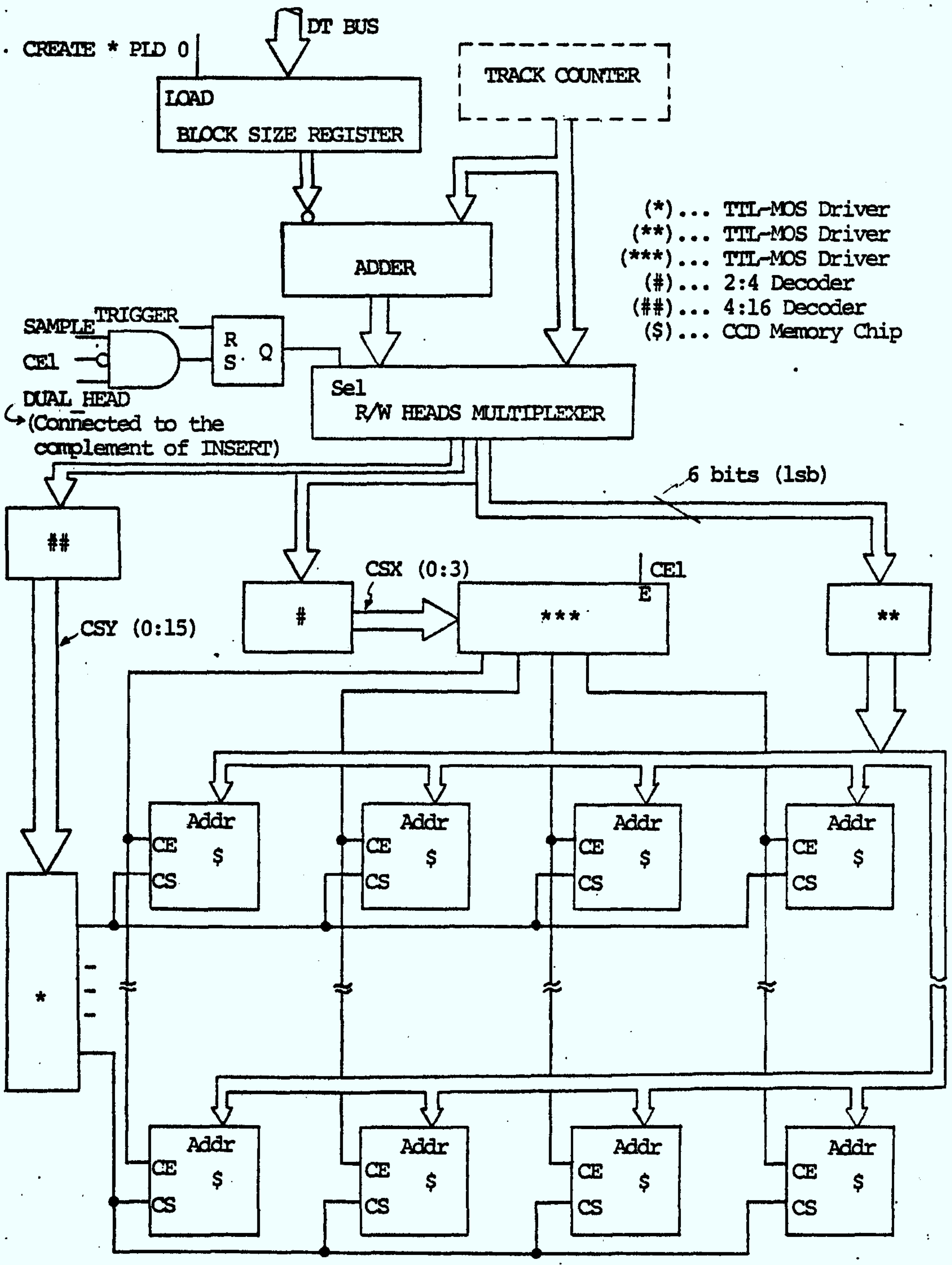
LOGIC DIAGRAM L-15

SYNCHRONIZER: SECTOR COUNTER, TRACK COUNTER, SPEED CONTROL



LOGIC DIAGRAM L-16

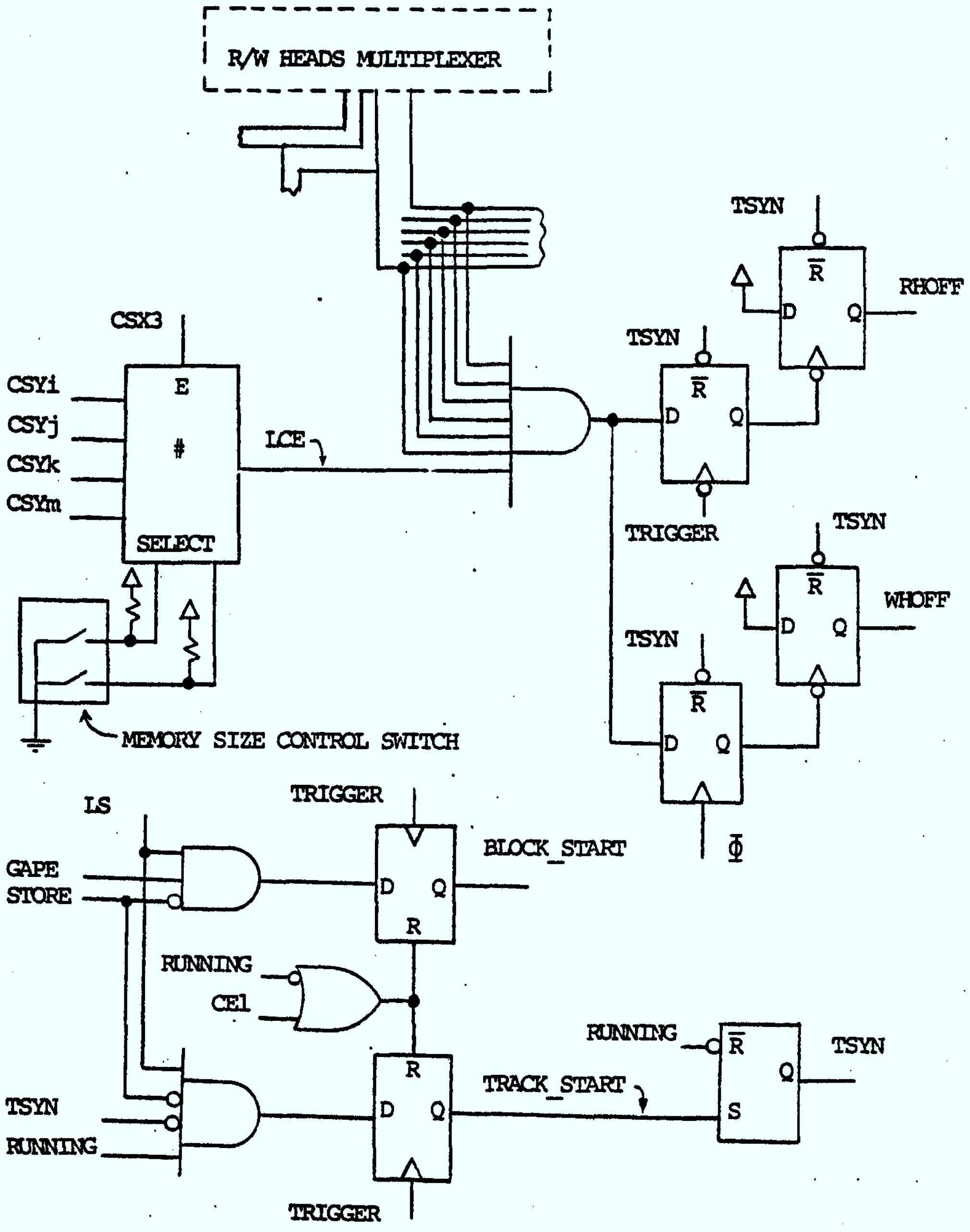
CCD TRACK: ADDRESSING FOR CCD CHIPS



(*)... TTL-MOS Driver
(**)... TTL-MOS Driver
(***)... TTL-MOS Driver
(#)... 2:4 Decoder
(##)... 4:16 Decoder
($)... CCD Memory Chip

LOGIC DIAGRAM L-17

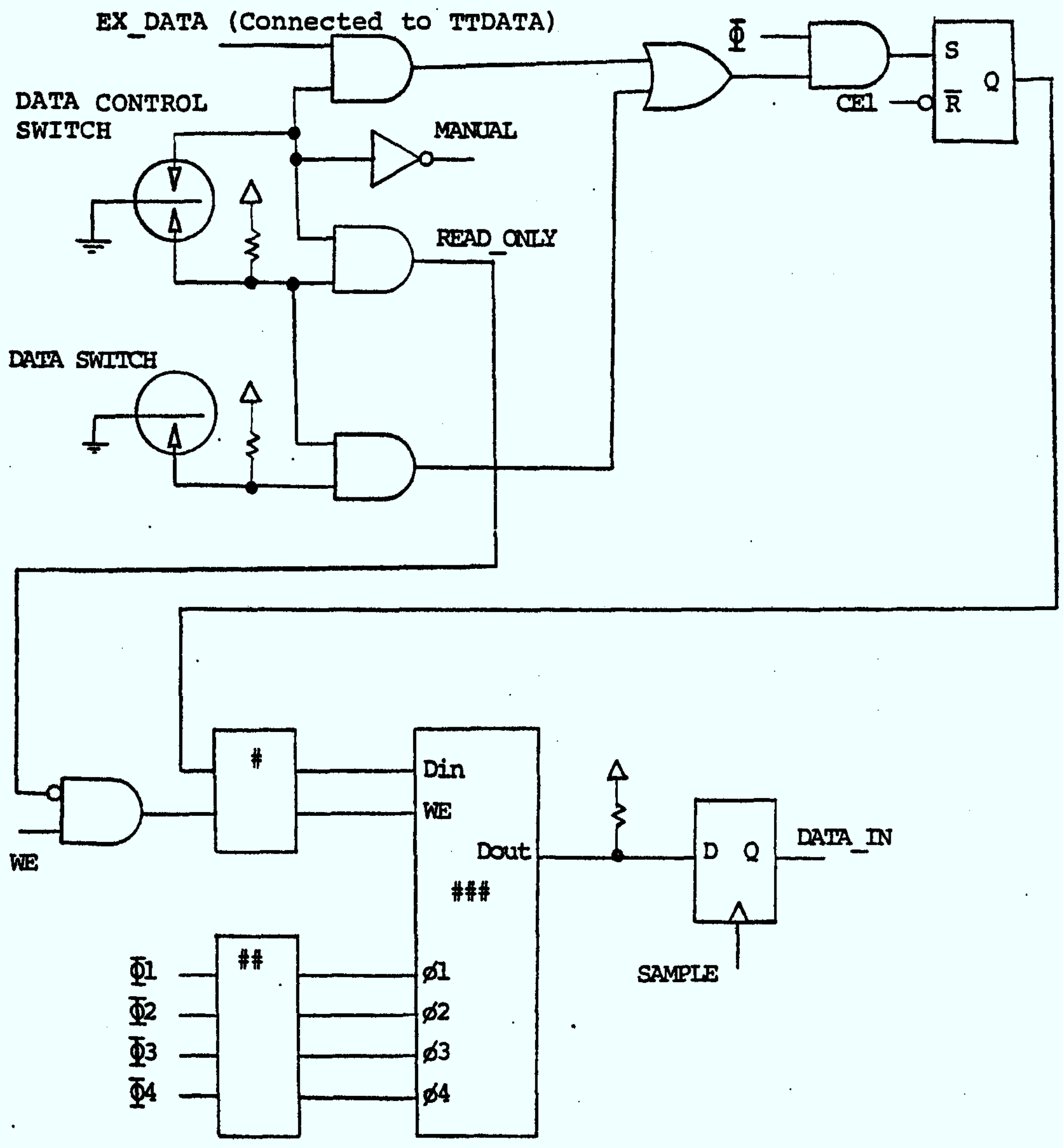
SYNCHRONIZER: BLOCK START, TRACK START, PHYSICAL END OF TRACK



(#)... 4:1 Multiplexer.

LOGIC DIAGRAM L-18

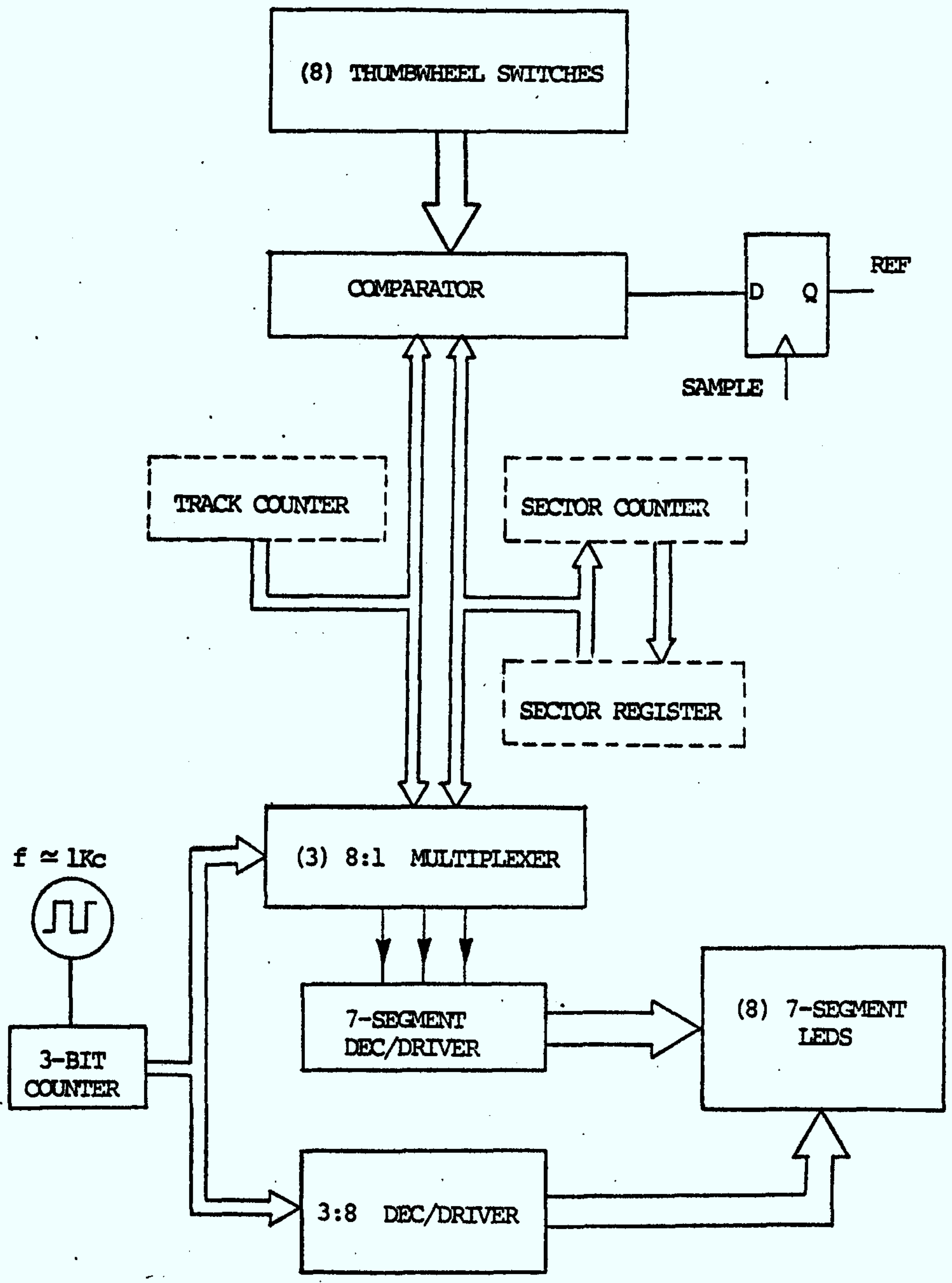
CCD TRACK: I/O, SHIFT DRIVE for CCD CHIPS



(#)... TTL to MOS Driver. The actual number of drivers depends on fan-out and on the physical distribution of CCD chips.
(##)... MOS Clock Driver. The actual number depends on fan-out.
(###)... CCD Chips. Logically, the pins shown above of all CCD chips are connected in parallel. But in practice, the limitations of drivers prevent that.

LOGIC DIAGRAM L-19

CCD TRACK: REF CONTROL, LED OCTAL DISPLAY



8.2 RAP 2 - System Characterization and Measurement

8.2.1 Overview

A prototype system incorporating a PDP 11/45 Main Frame, a PDP 11/10 Controller and two cells has been fabricated and tested in a variety of ways. The unifying theme in these tests has been the development of a set of 6 demonstration queries which characterize the system and its use. Section 8.2.5 provides a copy of these queries with printouts showing a tangible measure of intermediate and final results. In normal use, input in either SEQUEL commands or RAP Macro-Assembler language would be used. Output would be the same, display of intermediate results not being essential.

Section 8.2.2 provides a sofware/user characterization centering about the Block Diagram in section 8.2.2. Sizes of software resources are given, where fixed, on the diagram, or, where query-variable, in the table of Query Statistics in section 8.2.2.6 Various operation times are defined in 8.2.2.1 with results of measurement in 8.2.2.5.

A corresponding hardware system description is provided in section 8.2.3 with data on channel widths and rates provided.

The spirit of the whole presentation is to provide data in a modular form with enough discrete data points made available to permit estimation (at some effort) of the performance of alternate systems incorporating different processors, cell sizes, cell quantities, or other changes.

8.2.2 Software Characterization

8.2.2.1 Definition of Characteristic Times

Modules Involve

Compilation Time

- the time taken to translate a SEQUEL query into RAP intermediate code
- the SEQUEL code exists on a "s" file (with no comments) E
- the RAP intermediate code is written into a "r" file

Input Formatting Time

- the time taken to translate RAP intermediate code into RAP internal code F
- the RAP internal code is written to a "clt" file and is passed to the controller

Output Formatting Time

- the time taken to transform raw data that is retrieved from RAP into a displayable form
- the following important formatting is done:
  - relation and domain headings are produced J
  - integer domains are transformed from binary form into character form
- the retrieved data exists on a file names "raio 1"
- the formatted data is written on a "o" file
- deletion and updating do not require output formatting since no data is retrieved

Response Time

- the time taken to "execute" a query including:
  - RAP input formatting time
  - time taken to pass information to RAP from front-end F H J
  - actual RAP execution time
  - time taken to pass information to front-end from RAP
  - ouput formatting time

8.2.2.2 Major Program Steps

There are 4 possible steps of program execution:

1. Initialization of program.
2. Compilation of SEQUEL.
3. Execution of RAP
   - RAP Input Formatting (3a)
   - RAP Execution (3b)
   - Output Formatting (3c)
4. Termination

Steps 1, 4 must always be executed.

8.2.2.3 Procedures for Time Estimation

Compilation Time = Run (1 + 2 + 4) - Run (1 + 4)

Input Formatting Time = Run (1 + 2 + 3a + 4) - Run (1 + 2 + 4)

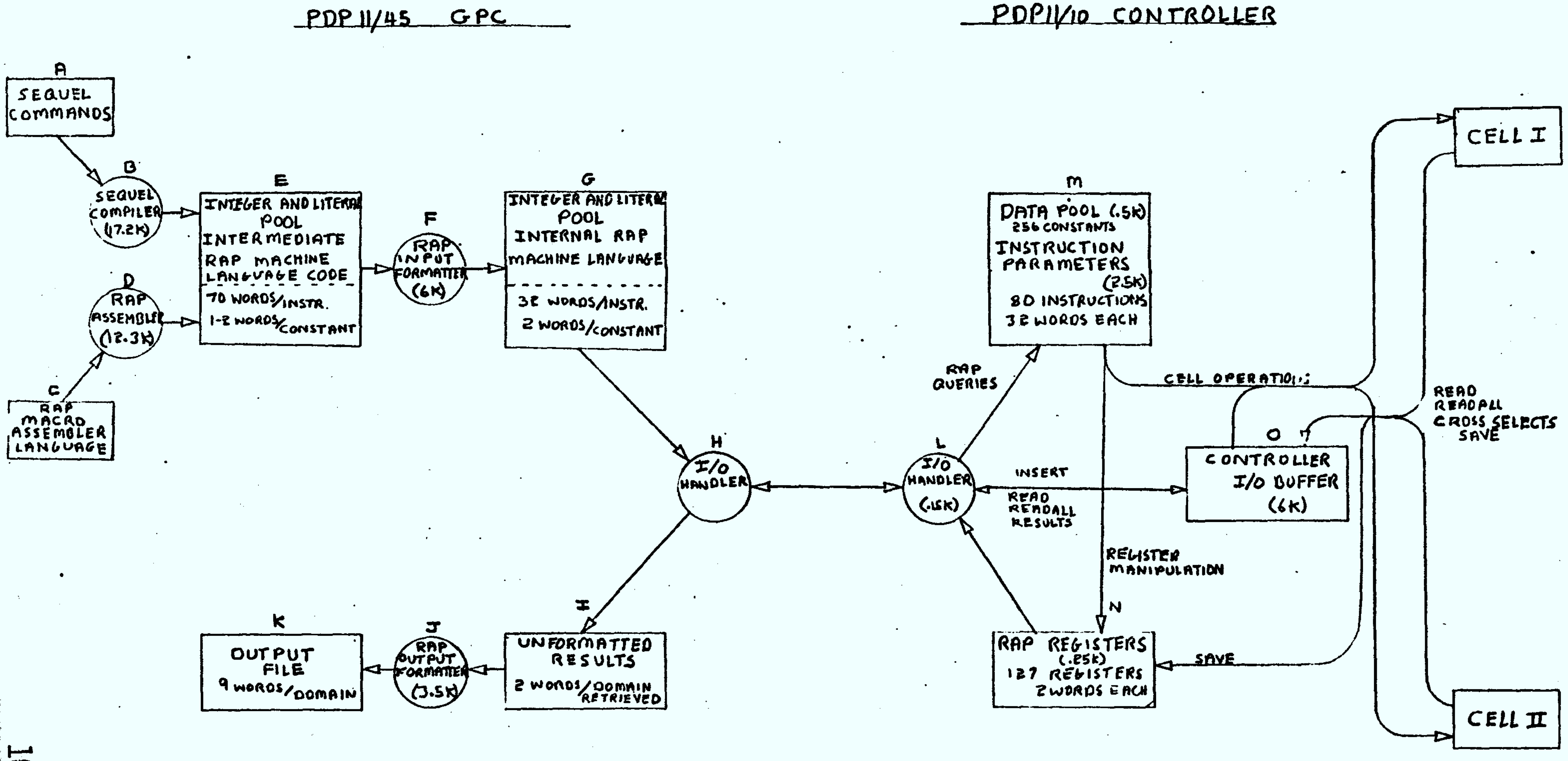
Output Formatting Time = Run (1 + 2 + 3a + 3b + 3c + 4) -
Run (1 + 2 + 3a + 3b + 4)

Response Time + Run (1 + 2 + 3a + 3b + 3c + 4) - Run (1 + 2 + 4)

8.2.2.4 SOFTWARE COMMUNICATION - BLOCK DIAGRAM

DIAGRAM I

SOFTWARE COMMUNICATION



8.2.2.5 EXECUTION TIME ESTIMATES OF RAP FRONT-END SOFTWARE

Query #	Type of Time	Compilation Time		Input Formatting Time		Output Formatting Time		Response Time	
		mean	sdev	mean	sdev	mean	sdev	mean	sdev
1	real	0.9	.697	1.1	.707	.4	.577	2.8	.577
	user	.11	.090	.03	.148	.03	.156	.07	.133
	sys.	.65	.246	.72	.222	.08	.206	1.44	.263
2	real	1.0	.721	1.4	.667	.4	.516	4.6	.516
	user	.08	.119	.02	.133	.12	.145	.20	.139
	sys.	.69	.261	.75	.260	.26	.344	3.35	.289
3	real	1.2	.721	1.7	.707	.4	.577	3.7	.707
	user	.19	.138	.03	.163	.06	.116	.10	.143
	sys.	.88	.370	1.02	.374	.09	.329	1.95	.429
4	real	1.2	.721	1.3	.606	0	0	2.4	.516
	user	.13	.076	.02	.111	0	0	.05	.114
	sys.	.90	.263	.77	.271	0	0	1.34	.273
5	real	.6	.503	.9	.316	0	0	2.4	.516
	user	.07	.070	.01	.111	0	0	.04	.145
	sys.	.43	.274	.72	.288	0	0	1.24	.307
6	real	1.5	.594	1.4	.577	.7	.707	4.4	.577
	user	.21	.143	.03	.158	.04	.152	.12	.165
	sys.	186	.367	.75	.350	.19	.245	2.55	.388

Note

All times are given in seconds. 10 trials were done for each mean. All were done on a multiple-user system occupied by only a single user in addition to systems functions. The latter account for some variation.

real	= actual elapsed time
user	= actual program execution time
system	= operating system time for file access

8.2.2.6 DEMONSTRATION QUERY STATISTICS

		SOURCE		COMPILATION		RETRIEVAL					
Query #	Type of Query	Sequel Source (words)	RAP Macro Assembler Source (words)	RAP Intermediate Code (words)	RAP Internal Code (words)	Tuples Retrieved	Data Retrieved (words)	Formatted Data (words)	RAP Revolutions	Execution Time for Controller and Cell (sec.)	CPU Time and File (Disc) Excess Time
1	Simple Retrieval	61	68	208	98	3	18	189	3	.07	2.27
2	Projection Retrieval	28	125	403	192	28	88	592	115	2.65	4.32
3	Nested Selection	95	185	669	322	11	66	405	13	.30	3.12
4	Update	73	116	414	200	0	0	0	5	.13	2.42
5	Deletion	25	17	141	66	0	0	0	1	.02	1.78
6	Physical Join	98	323	1263	610	14 (7+7)	56	391	63	1.37	3.74
	Diagram Label	A	C	E	G	Cell I,II	I	K	Cell I; II		

8.2.3 Hardware Characterization

8.2.3.1 Module Defintion

Modules Involved

UNIBUS U

- 56 lines (16 data, 18 address, 22 control)
- asynchronous operation with maximum data rate of 40 mbits/sec (both 11/45 and 11/10)
- all peripherial devices appear as memory locations
- DMA can be used by peripherials

DMA INTERFACE P, R

- DMA interfacrs on both buses allow memory to memory transfers in either direction
- datapath between P and R interface is 16 bits
- asynchronous operation with maximu of 20 mbits/sec
- data transfers possible are:

a)	block address	block address
b)	single address	single address
c)	single address	single address

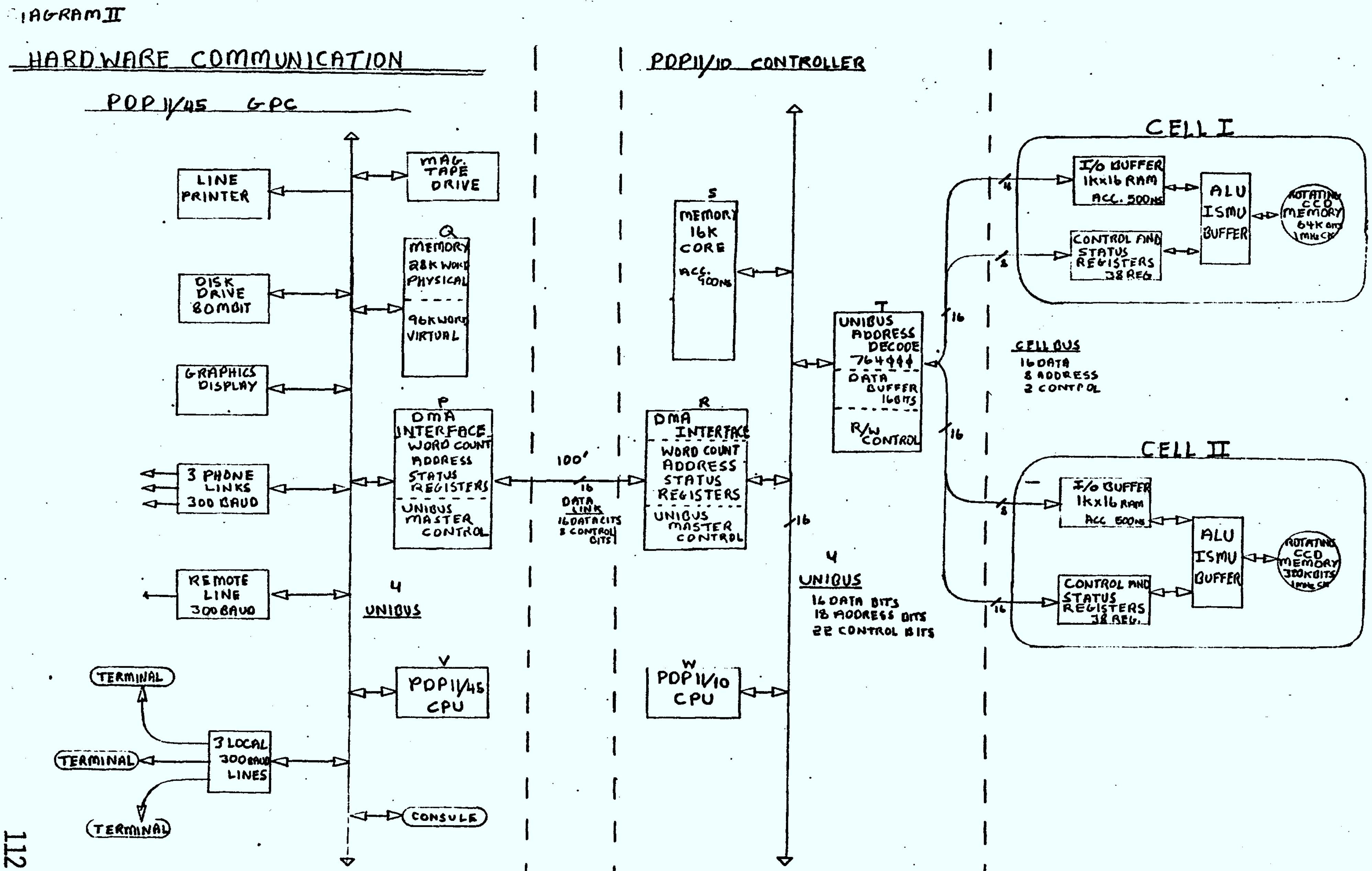
CELL INTERFACE T

- decodes a cell address block 764000 764777
- beuffers unibus from cell bus
- cells appear to be unibus memory locations

CELL I, II C

- I/O buffer is a 1K work two port ram in which retrieval data is written by the cell
- appears as a single memory location on unibus (memory address pointer can be manipulated)
- 38 registers for control and status of cell appear as memory locations on unibus

8.2.3.2 HARDWARE COMMUNICATION - BLOCK DIAGRAM



8.2.3.3. RAP - DMA INTERFACE RATES FOR READ AND/OR WRITE

including time to setup the slave device

Transfer Type	Total Bytes	Block Size Bytes	Number of Block Transfers	Time sec.	Rate Kbytes/sec.	Rate kbits/sec.
W	10^7	10^3	10^4	94.4	106	847
R-W	10^7	500	10^4	94.7	106	845
R-W	10^6	500	10^3	15.8	63	506
R-W	$8x10^5$	400	10^3	15.1	53	432
R-W	$6x10^5$	300	10^3	14.1	43	340
R-W	$4x10^5$	200	10^3	13.4	30	239
R-W	$2x10^5$	100	10^3	13.1	15	122
R-W	$2x10^4$	100	10^2	1.6	12	100
R-W	$2x10^5$	10	10^4	105	2	16

8.2.3.4 RELATIVE SPEEDS

PDP 11 FAMILY CALIBRATION

Register to Register Transfer Times as a Measure of Relative Speed

Machine	11/05	11/10	LSI 11	11/03	11/04	11/20	11/34	11/40	11/45	11/70
Time in usec	3.7	3.7	3.5	3.5	2.9	2.3	1.8	0.9	0.45	.3

Memory to Memory Transfer Times

PDP 11/45 Memory to Memory MOV	3.8 usec
PDP 11/10 Controller Memory to Cell Memory	MOV 7.4 usec

Demonstration of the

RAP System

Using the

SEQUEL Query Language

and the

Actual RAP Hardware

DRIVER

DOMAIN NAME	DOMAIN TYPE	DOMAIN LENGTH
DRIVER_NO	INTEGER	2 BYTES
SURNAME	CHARACTER	4 BYTES
INITIAL	CHARACTER	1 BYTE
HOME	CHARACTER	4 BYTES
SEX	CHARACTER	1 BYTE
MARITAL_STATUS	CHARACTER	1 BYTE
HEIGHT	INTEGER	2 BYTES
BIRTH_YEAR	INTEGER	2 BYTES
DAY_HIRED	INTEGER	1 BYTE
MONTH_HIRED	INTEGER	1 BYTE
YEAR_HIRED	INTEGER	2 BYTES
SALARY	INTEGER	2 BYTES

TRIP

DOMAIN NAME	DOMAIN TYPE	DOMAIN LENGTH
TRIP_NO	INTEGER	2 BYTES
ORIGIN	CHARACTER	4 BYTES
DESTN	CHARACTER	4 BYTES
LV	INTEGER	2 BYTES
AR	INTEGER	2 BYTES
TRAVEL_TIME	INTEGER	2 BYTES
MILEAGE	INTEGER	2 BYTES
FARE	INTEGER	2 BYTES
DRIVER_NO	INTEGER	2 BYTES

```
%RAP.SEQUEL

                  *** SEQUEL COMPILER - VERSION 4.1 ***

--->FILE 'QUERY1.S'

--->
%     QUERY 1

      RETRIEVE THE SURNAME, HOME CITY AND YEAR OF BIRTH OF THOSE DRIVERS
      WHO LIVE IN MONTREAL, AND WERE BORN AFTER 1940                          %

      +SELECT SURNAME, HOME, BIRTH_YEAR
       FROM   DRIVER
       WHERE  HOME = 'MONT'
         AND  BIRTH_YEAR > 1940+

THE SEQUEL STATEMENT HAS BEEN TRANSLATED IN 3 RAP INSTRUCTIONS

--->RAP

 1)     SELECT MARK(M1) [DRIVER:BIRTH_YEAR>1940 & HOME='MONT']
 2)     READ_ALL RESET(M1) [DRIVER(SURNAME,HOME,BIRTH_YEAR):MKED(M1)] [QUERY1.O]
 3)     EOQ

--->EXECUTE

QUERY TRANSMITTED

QUERY EXECUTION:
        3 REVOLUTIONS TO EXECUTE.
        4 SIXTIETHS OF A SECOND.

 3 TUPLES RETRIEVED

--->DISPLAY *

 +DRIVER                                                    +
 +                                                          +
 +SURNAME          +HOME            +BIRTH_YEAR             +
-------------------------------------------------------------
 +MARX             +MONT            +1947                   +
 +PIKE             +MONT            +1948                   +
 +COX              +MONT            +1953                   +

 --->FILE 'QUERY2.S'

 --->
 %     QUERY 2

       RETRIEVE THE ORIGINS, AND DESTINATIONS OF ALL OF THE DIFFERENT ROUTES RUN
       BY THE BUS COMPANY.                                                    %

       +SELECT UNIQUE ORIGIN, DESTN
        FROM   TRIP+

 THE SEQUEL STATEMENT HAS BEEN TRANSLATED IN 6 RAP INSTRUCTIONS

 --->RAP

  1)       SELECT MARK(M1M2) [TRIP]
  2)  L1   SAVE(1) RESET(M2) [TRIP(ORIGIN,DESTN):MKED(M2)] [REG(1)-REG(2)]
  3)       SELECT RESET(M1M2) [TRIP:MKED(M2) & ORIGIN=REG(1) & DESTN=REG(2)]
  4)       BC L1, TEST [TRIP:MKED(M2)]
  5)       READ_ALL RESET(M1) [TRIP(ORIGIN,DESTN):MKED(M1)] [QUERY2.O]
  6)       EOQ

 --->EXECUTE

 QUERY TRANSMITTED

 QUERY EXECUTION:
         115 REVOLUTIONS TO EXECUTE.
         159 SIXTIETHS OF A SECOND.

  28 TUPLES RETRIEVED
```

```
--->DISPLAY *

 +TRIP                              +
 +                                  +
 +ORIGIN           +DESTN           +
-------------------------------------
 +TORO             +LOND            +
 +LOND             +WIND            +
 +WIND             +LOND            +
 +LOND             +TORO            +
 +TORO             +HAM             +
 +HAM              +NF              +
 +NF               +HAM             +
 +HAM              +TORO            +
 +TORO             +MONT            +
 +MONT             +TORO            +
 +TORO             +KING            +
 +KING             +MONT            +
 +MONT             +KING            +
 +KING             +TORO            +
 +TORO             +PETE            +
 +PETE             +OTTA            +
 +OTTA             +PETE            +
 +PETE             +TORO            +
 +TORO             +OTTA            +
 +OTTA             +TORO            +
 +TORO             +BARR            +
 +BARR             +NBAY            +
 +NBAY             +BARR            +
 +BARR             +TORO            +
 +MONT             +OTTA            +
 +OTTA             +NBAY            +
 +NBAY             +OTTA            +
 +OTTA             +MONT            +




--->FILE 'QUERY3.S'

--->
%      QUERY 3

       CALCULATE THE AVERAGE AND TOTAL MILEAGE DRIVEN BY DRIVERS WHO LIVE
       IN TORONTO, AND RETRIEVE THE TRIP NUMBERS, ORIGINS, AND DESTINATIONS
       OF THEIR TRIPS.                                                       %


      +SELECT TRIP_NO, ORIGIN, DESTN, AVG(MILEAGE), SUM(MILEAGE)
       FROM   TRIP
       WHERE  DRIVER_NO IS IN +SELECT DRIVER_NO
                               FROM   DRIVER
                               WHERE  HOME = 'TORO'++

THE SEQUEL STATEMENT HAS BEEN TRANSLATED IN 10 RAP INSTRUCTIONS

--->RAP


 1)     SELECT MARK(M1) [DRIVER:HOME='TORO']
 2)     CROSS_SELECT MARK(M1) [TRIP:DRIVER_NO=DRIVER.DRIVER_NO] [DRIVER RESET(M1):MKED(M1)]
 3)     SUM [TRIP(MILEAGE):MKED(M1)] [REG(1)]
 4)     COUNT [TRIP:MKED(M1)] [REG(2)]
 5)     RDIV [REG(1)] [REG(2)]
 6)     READ_REG [REG(1)]
 7)     SUM [TRIP(MILEAGE):MKED(M1)] [REG(1)]
 8)     READ_REG [REG(1)]
 9)     READ_ALL RESET(M1) [TRIP(TRIP_NO,ORIGIN,DESTN):MKED(M1)] [QUERY3.O]
10)     EOQ
```

```
--->EXECUTE

QUERY TRANSMITTED

QUERY EXECUTION:
        13 REVOLUTIONS TO EXECUTE.
        18 SIXTIETHS OF A SECOND.

AVERAGE(MILEAGE) FROM TRIP IS 147

SUM(MILEAGE) FROM TRIP IS 1615

 11 TUPLES RETRIEVED

--->DISPLAY *

 +TRIP                                                   +
 +                                                       +
 +TRIP_NO           +ORIGIN            +DESTN            +
---------------------------------------------------------
 +101               +TORO              +LOND             +
 +106               +LOND              +TORO             +
 +201               +HAM               +NF               +
 +206               +NF                +HAM              +
 +300               +TORO              +MONT             +
 +400               +KING              +MONT             +
 +600               +TORO              +OTTA             +
 +601               +OTTA              +TORO             +
 +700               +TORO              +BARR             +
 +705               +NBAY              +BARR             +
 +705               +BARR              +TORO             +
```

```
--->FILE 'QUERY4.S'

--->
%     QUERY 4

      DUE TO INCREASING COSTS, THE FARE OF TRIPS BETWEEN TORONTO AND LONDON
      MUST BE RAISED TO 8 DOLLARS.                                              %

      +UPDATE  TRIP
       REPLACE FARE = 8
       WHERE   ORIGIN = 'TORO' AND DESTN = 'LOND'
          OR   ORIGIN = 'LOND' AND DESTN = 'TORO'+

THE SEQUEL STATEMENT HAS BEEN TRANSLATED IN 6 RAP INSTRUCTIONS

--->RAP

 1)     SELECT MARK(M1) [TRIP:DESTN='LOND' & ORIGIN='TORO']
 2)     SELECT MARK(M2) [TRIP:DESTN='TORO' & ORIGIN='LOND']
 3)     SELECT MARK(M3) [TRIP:MKED(M2) + MKED(M1)]
 4)     REPLACE RESET(M3) [TRIP(FARE):MKED(M3)] [8]
 5)     SELECT RESET(M1M2) [TRIP]
 6)     EOQ

--->EXECUTE

QUERY TRANSMITTED

QUERY EXECUTION:
        5 REVOLUTIONS TO EXECUTE.
        8 SIXTIETHS OF A SECOND.

 0 TUPLES RETRIEVED
```

```
--->FILE 'QUERY5.S'

--->
%      QUERY 5

       DRIVERS WHO LIVE IN MONTREAL GO ON A WILDCAT STRIKE. WITH RELUCTANT
       CONSENT FROM THE NONE-TOO-POWERFUL DRIVERS' UNION, MANAGEMENT
       PROCEEDS TO FIRE THESE RECALCITRANT WORKERS.                          %

       ←DELETE DRIVER
        WHERE  HOME = 'MONT'→


THE SEQUEL STATEMENT HAS BEEN TRANSLATED IN 2 RAP INSTRUCTIONS

--->RAP


 1)      DELETE [DRIVER:HOME='MONT']
 2)      EOQ

--->EXECUTE

QUERY TRANSMITTED

QUERY EXECUTION:
        1 REVOLUTIONS TO EXECUTE.
        1 SIXTIETHS OF A SECOND.


 0 TUPLES RETRIEVED
```

```
--->FILE 'QUERY6.S'

--->
%      QUERY 6

       FOR ALL DRIVERS WHO LIVE IN OTTAWA, GENERATE A LIST OF THE TRIPS
       DRIVEN BY EACH DRIVER, ALONG WITH THE DRIVERS' NAMES AND NUMBERS.     %

       ←JOIN ON TRIP.DRIVER_NO = DRIVER.DRIVER_NO
            ←SELECT DRIVER_NO, SURNAME
             FROM   DRIVER
             WHERE  HOME = 'OTTA'→
        WITH
            ←SELECT ORIGIN, DESTN
             FROM   TRIP→→

THE SEQUEL STATEMENT HAS BEEN TRANSLATED IN 19 RAP INSTRUCTIONS

--->RAP


 1)      SELECT MARK(M1M2) [DRIVER:HOME='OTTA']
 2)      SELECT MARK(M1) [TRIP]
 3)      BC L1, TEST [DRIVER:MKED(M1)]
 4)      SELECT RESET(M1) [TRIP]
 5)      BC END
 6) L1   BC L2, TEST [TRIP:MKED(M1)]
 7)      SELECT RESET(M1M2) [DRIVER]
 8)      BC END
 9) L2   SAVE(1) RESET(M1) [DRIVER(DRIVER_NO):MKED(M1)] [REG(1)]
10)      SELECT MARK(M2) [TRIP:DRIVER_NO=REG(1) & MKED(M1)]
11)      BC L3, TEST [TRIP:MKED(M2)]
12)      BC L4
13) L3   READ [DRIVER(DRIVER_NO,SURNAME):UNMKED(M1) & MKED(M2)] [APPEND QUERY6.O]
14)      RETRIEVE(1) RESET(M2) [TRIP(ORIGIN,DESTN):MKED(M1) & MKED(M2)] [APPEND QUERY6.O]
15)      BC L3, TEST [TRIP:MKED(M2)]
16) L4   SELECT RESET(M2) [DRIVER:UNMKED(M1) & MKED(M2)]
17)      BC L2, TEST [DRIVER:MKED(M1)]
18)      SELECT RESET(M1) [TRIP]
19) END EOQ
```

```
--->EXECUTE

QUERY TRANSMITTED

QUERY EXECUTION:
             63 REVOLUTIONS TO EXECUTE.
             82 SIXTIETHS OF A SECOND.

 7 TUPLES RETRIEVED

--->DISPLAY *

 +DRIVER                                            +TRIP                                +
 +                                                  +                                    +
 +DRIVER_NO          +SURNAME                       +ORIGIN         +DESTN               +
---------------------------------------------------------------------------------------------
 +103                +WATT                          +TORO           +LOND                +
 +103                +WATT                          +LOND           +TORO                +
 +106                +BEGG                          +OTTA           +PETE                +
 +106                +BEGG                          +PETE           +TORO                +
 +118                +BOND                          +NBAY           +OTTA                +
 +121                +BARR                          +WIND           +LOND                +
 +133                +CARR                          +OTTA           +NBAY                +

--->QUIT
```

ENGINEERING
EO 8.2.5

FACULTY OF APPLIED SCIENCE & ENGINEERING • UNIVERSITY OF TORONTO

JUNE 1976 No. 30

DATA MANAGEMENT BY RAP

A NEW COMPUTER FOR DATA BASE MANAGEMENT

Recent developments in semi-conductor technology have brought about a great change, not only in traditional computing machinery but in what is economically conceivable in new computing organizations. One result is the RAP machine which may likely revolutionize the way people utilize data processing.

DATA BASE MANAGEMENT SYSTEMS

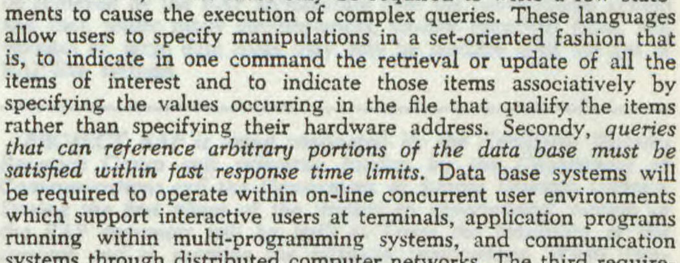
A **data base** can be thought of as a structured collection of commonly pooled data which is accessible by concurrent users through computer systems. Data bases constitute the heart of management information systems. These systems comprise the total organizations of people and computer technology for the purpose of providing *planners, managers, and researchers* the timely and reliable data on which they can base decisions and inferences. Applications of data bases and management information systems span all aspects of society; for example, *economic monitoring, regional planning, political forecasting, hospital administration, and business* to name but a few. Traditional computer approaches to implementing earlier versions of data bases were to provide users a collection of **file processing programs.** The data for the applications were stored in computer files on secondary storage devices and were directly manipulated by programs of the application. Applications implemented this way were intimately tied to the minute details of computer operations and specific storage devices. As more applications were demanded, the number of specialized files increased, causing many problems. Some of these files often contained redundant information in different formats, causing the values of an item to become inconsistent since some files are updated by differing procedures on varying time-scales. Also, any variations in the data organization require extensive changes to the programs that access the data and vice versa. The possibilities for on-line or real-time simultaneous access to common data by users was negligible and forced the costs of data processing software to soar. To alleviate this undesirable trend, the concept of **data base management** systems evolved. These systems aim at creating data bases that have an existence separate from the specific applications using them. They are complex integrations of computer software and hardware which attempt to provide their users with a "logical view" of a data base that will insulate users from the details of data storage and manipulation. A **query language** is also available by which users can readily specify the retrievals and updates of the data stored according to the "logical view". A data base management system, by controlling and monitoring access to a data base, can also promote the security and consistency of the data and eliminate data duplication.

THE RELATIONAL VIEW OF DATA

To understand data bases better, consider one of the newest and most important logical views of data bases; the **Relational Model of Data.** A relation can be viewed simply as a table of data whose rows contain information which describe a set of similar entities such as persons, places, or things. Figure 1 shows a collection of relations which describe a department store. The name of the table or relation identifies the set of entities being described such as EMPLOYEE, SALES, and LOCATION. The names of the columns identify the attributes which are used to describe the characteristics of each entity for example NAME, SALARY, and DEPARTMENT of an EMPLOYEE. Each row contains a set of values — one for each attribute — which characterizes a particular entity for example, JOHN SMITH, $20,000.00, and men's SHOES. The order in which rows or columns occur is immaterial to the users logical view of their relations. A relational data base is composed of a collection of time-varying relations which may change because of modifications, insertions, and deletions. Data in two or more relations can be interrelated through common attributes which appear in each of the relations. This allows users to execute queries which have complex selection criteria. For example, to find all the employees who work on the second floor, the location relation can give a list of departments on the second floor. These departments can then be used to select the employees who work in them. (Fig. 1) The **Relational Model** provides users a view of data that is simple, consistent, and yet computationally complete with respect to data processing requirements. Other types of logical data base views, such as hierarchies and networks, can be constructed from relations. Languages for manipulating relational data bases are simple but powerful. The relational model of data and its associated languages can bring data base capabilities to casual, clerical, and technical users who are not computer specialists.

THE LIMITATIONS OF CONVENTIONAL COMPUTERS

The full potential of data bases will be realized only if three important requirements are implemented. First, *the languages provided to users must be sufficiently user-oriented and powerful enough to permit simple specifications of desired data manipulations.* That is, a user must only be required to write a few statements to cause the execution of complex queries. These languages allow users to specify manipulations in a set-oriented fashion that is, to indicate in one command the retrieval or update of all the items of interest and to indicate those items associatively by specifying the values occurring in the file that qualify the items rather than specifying their hardware address. Secondy, *queries that can reference arbitrary portions of the data base must be satisfied within fast response time limits.* Data base systems will be required to operate within on-line concurrent user environments which support interactive users at terminals, application programs running within multi-programming systems, and communication systems through distributed computer networks. The third requirement has to do with the *technical administration of data bases.* The separation of the physical data from its users causes the responsibility for efficient performance to be transferred from the user to the administrator (collectively called the Data Base Administrator) of the system. The responsibility for system tuning was originally distributed over several users and file processing systems. Today, the data base administrator must make decisions,


Hardware Being Developed for the RAP Prototype

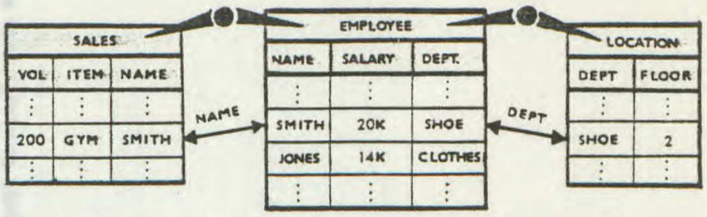
often conflicting with individual user requirements for all the users. Therefore, it is imperative that the tuning of data base systems be accomplished effectively and easily, since several reliability and cost problems arise when using conventional computers to implement modern data management systems. The aim of the University of Toronto's Project RAP is to provide new computer architecture exploiting new semi-conductor technology to satisfy these requirements.

Since conventional machines lack the set-oriented processing and associative addressing architecture required by modern data bases, implementors must simulate this architecture through costly software. Accordingly, large complex programs must be supplied that map the user's view into the physical reality of the machine and provide access paths to permit fast location of arbitrary portions of the data base. Access paths are extra data and/or sorting strategies which "index" the original data base by providing a more direct access to a specific item's storage location. Additional software must be provided to utilize and maintain the access paths. Today's machines with their need for costly software to provide access paths and logical to physical mappings are an attempt to force a machine to do a job for which it was not designed and places a tremendous overhead on conventional computer systems. For example, access paths provide fast retrieval of data at the expense of slower updates. This happens because updates to the data base must also be reflected in the access paths. This trade-off is just one difficulty in administering data base due to the dynamic aspects of user environments.

THE RAP APPROACH TO DATA MANAGEMENT

The solution to these problems is the elimination of the need for access paths and mappings. This can be done by developing new computers whose architecture utilizes many processors and memories in parallel and which address data associatively. The data can then be divided into smaller more processable segments and distributed across many processors and memories to be searched and manipulated simultaneously. Associativity helps eliminate access paths and is logical to physical mappings. Parallelism causes associative addressing and set-oriented processing to be performed at high speeds. This new approach to data base system implementation has recently become feasible because of new developments in semi-conductor technology where extensive amounts of memory can be miniaturized and entire computers called microprocessors can be squeezed into the space of tiny low cost silicon chips.

The **Relational Associative Processor,** RAP, being designed and implemented at the University of Toronto is based on these principles and technologies. RAP is organized to augment a conventional computer in order to support the implementation of efficient data base management. The design employs hundreds of adjacently connected processors and memories called **cells** which address data associatively. A statistical arithmetic unit is provided to calculate summary statistics. The cells and statistical unit are driven in parallel by a central controller. This organization is shown in Figure 2. Each cell is composed of a microprocessor specially designed for data management operations and a sequential circulating memory, a track of a drum or disk, charge-coupled device (CCD), bubble memory, or other. Each data base operation is executed in parallel within cells which operate directly on the data as it circulates through each cell processor. RAP provides an intermediate-level view of data and a collection of set-oriented instructions implemented entirely by hardware. The data organization is sufficiently general to support set-oriented operations on the commonly desired high-level user views of data: *hierarchical, network, and relational.* The format for each hardware instruction reflects the general structure of a data base query, that is, it specifies the operation to be performed, the items to be operated upon, and the criteria which associatively selects or addresses the portion of the data base for which the query applies. Some of the operations that can be performed are *complex selection, statistical calculation, retrieval, update, and data base creation, insertion, and deletion.*



SALES		
VOL	ITEM	NAME
:	:	:
200	GYM	SMITH
:	:	:

EMPLOYEE		
NAME	SALARY	DEPT.
:	:	:
SMITH	20K	SHOE
JONES	14K	CLOTHES
:	:	:

LOCATION	
DEPT	FLOOR
:	:
SHOE	2
:	:

FIG. 1 — *Several Relations that Describe a Department Store*

The processor has been designed to close the gap between the user's logical view of data and the way it is represented in the storage of the computer which supports the processing of the data. This will allow data base management system applications to be implemented more quickly because the implementor will no longer be concerned with the details of representing and searching a data base. Because the device more closely represents the users' view of a data base management system, data base queries can be formulated from just a few hardware instructions — often only a single instruction is sufficient. The RAP system is designed to execute the most important instructions within one simultaneous rotation of the cell memories. Studies have been conducted to compare the hypothetical performance of using RAP relative to using a conventional computer system for implementing a relational data base. Both approaches were modeled analytically. The results show that significant gains in query execution speed can be achieved by the RAP architecture over the conventional system and that, under many circumstances, on-line retrievals from, and updates of, large data bases may only be possible with the use of RAP-like systems.

PROJECT RAP is being conducted by the Computer Systems Research Group and primarily funded by the Department of Communications and the Department of Supply Services, Canada. During the first phase of the project an experimental version of the RAP computer will be built to prove the feasibility of the concept and to examine the software implications. When completed, it is hoped that the interest of potential users and manufacturers of computers will encourage the pursuit of a full scale prototype of this system which would ultimately lead to a marketable product.

The experimental version of RAP will consist of two cells with CCD memories and a controller capable of executing a third to half of the instruction set. Some of the hardware being developed for the prototype is shown in the photograph. A working system is expected to be ready by the fall of 1976. This will include software support for both a high-level query language and RAP assembler language interfaces.

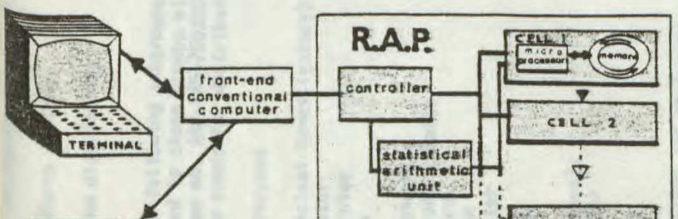


FIG. 2 — *Architecture of the RAP Processor*

KENNETH C. SMITH is a Professor of Electrical Engineering and Computer Science and Chairman of the Department of Electrical Engineering, STEWART SCHUSTER is jointly appointed to the Department of Computer Science and the Faculty of Management Studies, ESEN OZKARAHAN is a Visiting Assistant Professor of Computer Science from the Middle East Technical University of Turkey, appointed to the Department of Computer Science. All are members of the Computer Systems Research Group which is an interfaculty organization sponsored by the Department of Computer Science, the Faculty of Arts and Science and the Department of Electrical Engineering of the Faculty of Applied Science and Engineering at the University of Toronto.

ENGINEERING FORUM is made available to high school science and technology departments, and to graduates of the Faculty. Others who wish to receive ENGINEERING FORUM may be included in our mailings by writing to: Liaison Office, Faculty of Applied Science and Engineering, University of Toronto, Toronto, Ontario, M5S 1A4.

From: S. Schuster

Date: March 2, 1976

Re: Memo distribution

The following subgroups exist within the RAP project. Memo's addressed to these groups will be distributed their respective members (see memo no. RAP-SAS-760302-01-010 for reference nos.) Otherwise list the names for distribution.

RAP: everyone

PI (principal investigators):
Schuster
Ozkarahan
Smith

HW: (hardware):
Ozkarahan, Smith
Pereira
Nguyen
Schuster
Hunter
Huwito

SW (software):
Schuster
Chan
Tsonis

MA (modelling and analysis)
Ozkarahan
Schuster
Sevcik

IL (Industrial Liason):
Smith
Schuster
Ozkarahan

From: S. Schuster

Date: March 2, 1976

Re: revised memo reference numbers - for project work book

We continue to encourage the drafting of memos, working papers, etc. (not necessarily to be typed) for the project files and work books. Please use the following reference no system on all documents

1) ref# RAP-ABC-YYMMDD-##-TTT-(X)

where	RAP	is the project id
	ABC	are your initials
	YY	is the year
	MM	is the month
	DD	is the day
	##	is the index indicating the order in time for memos generated on the same day
	TTT	is the reference to the task numbers (see below)
	X	is the total number of pages
	[illegible]	[illegible]

Number all pages by x/X if X>1, where x is the page number and X is the total number of pages

2)

TTT	task
000	project descriptions (e.g. propaganda, layman descriptions)
005	progress reports
010	project organization
020	personnel and equipment procurment
030	general implementation memos (i.e. schdules, notes, etc)
040	hardware implementation documentation
050	other RAP specific memos (e.g. reliability, performance tradeoffs, extensions)
060	uses of RAP (e.g. networks, intelligent terminal, data models index processor)
070	Performance Comparison
080	Virtual memory RAP
100	RAP/SEQUEL Compiler
110	Instruction Simulator
120	Software Driver
130	Industrial/User Liason
140	Semi-Conductor Industry
150	Patents

PRODUCT PROPOSAL:

A DATA BASE COMPUTER

intel memory systems

A DIVISION OF INTEL CORPORATION
1302 N. Mathilda, Sunnyvale, CA 94086 • (408) 734-8102

Intel Corporation
Memory Systems Division
Sunnyvale, California

1. Introduction

Intel is investigating the possibility of developing a data base computer (DBC) which includes both novel hardware and complete data base management system software. The DBC will function either as a stand-alone application dedicated machine accessible through intelligent terminals or as a 'back-end' machine attached to medium and large host mainframes. A DBC consists of a conventional minicomputer and large capacity disks augmented by a special purpose processor and memory. The entire system will alleviate traditional indexing requirements yet achieve retrieval and update speeds not possible with current approaches. Other advantages of this approach are the extension of mainframe capability through off-loading, enhanced security and reliability, multi-mainframe sharing of the same data base, and simplified data base administration.

The DBC will provide users a high-level set-oriented data manipulation language offering non-procedural specification of data management tasks. It is anticipated that the system will be designed to support the relational model of data, although the special purpose hardware could provide hierarchical and network data bases. The system will be most cost-effective in environments requiring complex searches, unanticipated searches, interactive query processing, and/or in applications exhibiting high update traffic.

This document provides a summary of the DBC architecture and associated software facilities. Also included is a list of the potential advantages the computer offers data base management system users. It concludes with a summary of cost savings, performance improvement, and expected price of the Intel DBC.

2. The Data Base Computer

2.1 Hardware

2.1.1 Overview

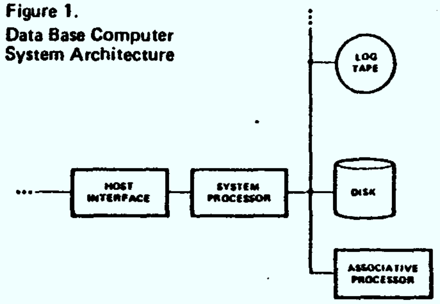
Intel's DBC is a simple organization of four types of components: a system processor, an associative processor, an array of conventional large capacity secondary storage devices, and a host interface. The basic architecture is shown in Figure1. The system processor is a medium to large minicomputer with customized software for coordinating functions of the data base management system (DBMS). The associative processor is a special purpose peripheral device designed to augment the relatively small system processor by implementing many data base operations in dedicated hardware. At any point in time the data base is partitioned and distributed between the associative processor memory and conventional disk storage. Migration of data between the associative processor and disk is dynamicly controlled by the DBMS software, and is transparent to the user. Tape is provided for bulk loading and for back-up and recovery logging.

The DBC will link to user machines through the host interface. This can be either a communications processor allowing very flexible access and multiple host sharing of the DBC or a high speed channel adaptor linking the DBC to a single host. Direct attachment of terminals to the DBC is also available.

2.1.2 The Associative Processor

The associative processor is the key element in making a feasible high performance DBC which provides fast retrieval and update of large data bases in response to high-level data manipulation commands. The associative processor's architecture is based on two facts. First, many data base operations are inherently set-oriented, requiring execution of the same operations on many records of the same file. Secondly, to achieve high data independence, data base addressing should be accomplished associatively by restricting user reference to data element content rather than position or location. Existing and newly developing data base systems simulate these desirable features on conventional computers via software thereby incurring tremendous

Figure 1.
Data Base Computer
System Architecture



exploits these features by incorporating special purpose parallel and associative hardware in its architecture.

The basic architecture of the associative processor, shown in Figure 2, consists of a controller and set of parallel cells. Each cell contains a processor and large capacity memory. The portion of the data base residing on the associative processor is distributed across the cell memories. The cell memories are constructed from CCD's, very high density, state-of-the-art, low cost semiconductor components. Each cell processor is identical, dedicated to one cell memory, and specifically constructed to execute high-level data definition and manipulation instructions. The controller of the associative processor receives program instructions from the system processor, decodes each instruction, and broadcasts control and data sequences to each cell. Each cell processor independently executes the instruction over the contents of its memory. Cell instructions exist for data definition, complex boolean selection, insertion, deletion, numeric or replacement update, retrieval, and computation of summary statistics. The number of cells can be varied as user requirement dictate.

The logical format of data stored on the associative processor is automatically interpreted by hardware. Direct execution of high-level instructions by the associative processor eliminates much of the software needed to provide a complete DBMS. In addition, the associative processor eliminates the need for traditional indexing software and data for performing searches over on-line data bases. By eliminating index data, the system can achieve balanced and fast response to complex selection, retrieval, and update operations and provide a greater potential for concurrent usage. Other benefits will be summarized later.

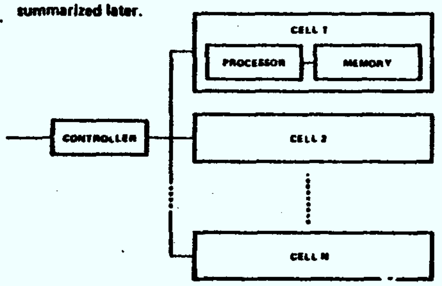


Figure 2. Associative Processor Architecture

2.2.1 Data Model

The purpose of a DBMS is to create a common pool of data decoupled from application programs and to give application programmers a logical model of the data base that is independent of device or physical organization details. Commercial systems today provide users one of three data models: hierarchical, network, or relational. The Intel DBC could be programmed to provide DBMS supporting any of these models. However, the naturally non-procedural, associative, and set-oriented characteristics of the relational model of data best suit the DBC architecture.

2.2.2 Access Methods

To provide fast access to arbitrary portions of the data base, the DBMS uses a combination of partitioning and migration strategies to allocate data between disk and associative processor storage. The DBMS uses parameters supplied by data base administrators and/or obtained through usage histories to place the most searched or updated data elements on the associative processor. Figure 3 depicts how data bases can be partitioned. A search request is executed by first accessing the associative processor. If further search is required over the disk portion to complete the data manipulation command, record id's are retrieved from the associative processor and used to access the disks' data. The search and migration of data between disks and the associative processor is, of course, transparent to the application programmer.

2.2.3 Data Definition, Manipulation, and Protection

A complete set of data definition, manipulation, and protection facilities will be provided. Data definition includes schema, subschema and derived schema definitions. Schema utilities such as data dictionary query, macro definition, and accounting are also to be provided. Performance monitoring information will be available through schema control.

The data manipulation language will provide high-level structured English syntax for specifying complex boolean and nested selections including statistical calculations, projection of data elements, joins, set construction or comparisions, and grouping. Selected records and data elements can be updated, inserted, deleted, sorted and retrieved. The manipulation statements can be embedded in programs running on host machines as parameters of CALL-statements or used as an interactive language through terminals hooked directly to the DBC.

integrity and recovery will be provided. Concurrency control will syncronize multithreaded query execution by providing the proper serialization of concurrent retrieval, but mutually exclusive update, while maintaining a high-level of data base consistency. Security and integrity mechanisms protect against unauthorized or invalid disclosures or updates. A back-up and recovery subsystem will produce the necessary journals and images for efficient and reliable recovery, roll-back, and roll-forward.

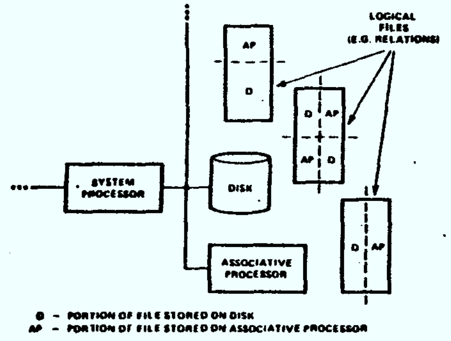

LOGICAL FILES (E.G. RELATIONS)
AP
D
D AP
AP D
SYSTEM PROCESSOR
DISK
ASSOCIATIVE PROCESSOR
D AP
D – PORTION OF FILE STORED ON DISK
AP – PORTION OF FILE STORED ON ASSOCIATIVE PROCESSOR

Figure 3. Data Base Partitioning

3. Potential Benefits

The potential benefits that can be incurred from the Intel DBC are grouped by the three principle features of the product: a distributed back-end system, providing a high-level set-oriented data manipulation language interface, and augmented by an associative processor. Each feature in the list enhances the capabilities and performance of the previous.

Back-End:

1) Extends host system capabilities by off-loading data base management functions to a separate system running in parallel.
2) Permits several possibly dissimilar host machines to share a common data base.
3) Provides a basis whereby dissimilar hosts on a network can exchange data via the DBC's logical data model.
4) Provides a basis for graceful conversion from one mainframe to another vendor's mainframe.
5) Provides the data base administrator distinct hardware, software, and data resources to tune without having to compromise or conflict with other mainframe users.
6) *applications via specialized hardware and customized* software.
7) Enhances data base security because of hardware separation of data base functions from the host operating system.
8) Reduces dependency of data base systems on host operating system or compiler changes.
9) Greater availability of services because distributed processing allows host or back-end availability when one or the other fails.

Set-Oriented Processing:

1) Reduces application programming, testing, and maintenance time. High-level commands are easier to code, more understandable because their semantics are closer to the end user's job, and are more concise statements of a user's requirements.
2) Reduces the amount of data transfer and hence reduces data communication requirements between the host and back-end. The host sends less data in the form of data manipulation commands when commands are expressed in high-level set-oriented languages. The back-end responds with less retrieved data. Only those data elements from the required records are returned to host.
3) Permits the DBMS to optimize the execution of a larger portion of the data manipulation requirement. Commands expressed in high-level languages permit the system, in advance, to analyze operations which affect sets of records. This permits the system to choose optimal access paths, buffering, and concurrency schemes because it can see a global picture of user requirements. This is not possible using record-at-a-time navigation oriented languages.

Associative Processor:

1) Provides very efficient execution of set-oriented commands because its associative and parallel architecture is oriented to processing several records at the same time.
2) Reduces and balances update and retrieval response times without traditional indexing methods and data.
3) Reduces total storage requirements because duplicated indexing data is eliminated.
4) Promotes greater concurrency because peripheral data structures such as indices are eliminated and therefore not subject to locking overhead.
5) Reduces data base administration complexity because simplified physical data organizations are achieved through hardware replacement of indices.
6) Reduces data manipulation command translation

user languages and data models.

7) Allows low cost incremental system upgrade because of the highly modular architecture of the associative processor.

8) Increases reliability by exhibiting graceful degradation due to the modular and distributed architecture of the associative processor.

9) Enhances the potential of the DBC to respond to complex security and integrity testing because of its ability to efficiently execute complex boolean tests over data bases.

10) Reduces the amount of data transferred between the associative processor and system processor because the associative processor responds to high-level commands and transfers only the required data items of qualified records. This is analogous to the advantages that set-oriented processing achieve for host and back-end communications.

4. Performance, Cost and Price

The Intel DBC is expected to be available fourth quarter, 1979. At this time, exact figures have not been determined for performance, and user application costs and price of a DBC required to meet the needs of a particular user. Instead, we first present a preliminary quantitative estimate followed by a qualitative summary of the potential DBC benefits that would directly reduce the total data base application cost/performance ratio. We conclude with price estimates for various components and example configurations of the DBC.

An analysis has been made to determine the execution times of various data manipulation functions when data resides entirely on the associative processor. Results show execution from 10 to 1000 times faster than conventional indexed implementations of the same functions. The most dramatic improvements are found in update operations. This massive processing power gives substantial lattitude for performance improvements in systems that incorporate large disk resident data bases. We conservatively estimate that the DBC would improve execution times and through-put by factors of 5 to 10.

1) Faster retrieval, update and reorganization response times due to specialized hardware and less complex software.

2) Greater through-put due to increased concurrency potential and faster response times.

3) Faster and more maintainable application software because of the high-level data manipulation language interface which provide a high degree of data-independence.

4) Greater potential for data base sharing by multiple and dissimilar hosts ranging from large mainframes to microprocessor based intelligent terminals.

Cost:

1) Frees-up costly large host CPU cycles which extends host life expectancy and/or increases work-load capability.

2) Reduces data base administration complexity and time because of the DBC's simple dedicated hardware and software architecture.

3) Lower cost upgrades because of the highly modular architecture of the DBC.

4) Reduces disk storage requirements and its associated maintenance costs because the DBC eliminates index data overhead.

5) Lower application development and maintenance costs through the use of a high-level data manipulation language.

Price:

System Processor = from \$30K to \$200K.
Disk = \$45K controller + \$25K per 200 MB (up to 1600 MB per controller).
Associative Processor = \$50K + \$6K per MB.
Tape = \$12K controller + \$14K per transport.

		SMALL	MEDIUM	LARGE
Basic DBC System	System Processor	$30K	$100K	$200K
	Associative Processor and Storage	10MB	50MB	100MB
	Associative Processor Price	$110K	$350K	$650K
	Software Price	$50K	$50K	$50K
	TOTAL	$190K	$500K	$900K
Secondary Storage Moved From Front End	Disk Storage	200MB	1000MB	2000MB
	Disk Price	$70K	$145K	$290K
	Tape Transports	1	2	4
	Tape Price	$26K	$40K	$68K
	TOTAL	96K	$185K	$358K

EXAMPLE SYSTEMS

9.3 Photographs

9.3.1 RAP Cells and Memory

9.3.2 RAP Controller - PDP 11/05

9.3.3 Host Machine - PDP 11/45

9.3.4 RAP II - Cell Boards
- ALU
- OPCODE III
- FORMATT I
- FORMATT II
- I/O BUFFER
- CONTROLLER I
- CONTROLLER II
- DATA MANIPULATOR

9.3.5 RAP II - Memory Boards
- Controller (For 1.5M Bits)
- Memory (320K Bits)

9.3.6 RAP II - Interfaces
- Cell to Controller
- Controller to Host

9.3.7 Some of the RAP team (from the left Nguyen, Pereira, Patkow, Hudyma, Smith, Soong, Schuster Radacz, Hawkins, Klebinoff).