RAP—An associative processor for data base management

by E. A. OZKARAHAN, S. A. SCHUSTER and K. C. SMITH
University of Toronto
Toronto, Ontario

INTRODUCTION

Problems with DBMS on conventional machines

Recent concepts in data base management systems (DBMS) necessitate making the logical view and the physical representation of data distinct from each other. Currently, this requirement has to be realized in the environment of conventional Von Neumann architecture. This creates the need for several levels of indirection for mapping one structure into the other. Also, efficient search mechanisms are needed to handle large data bases within concurrent processing and on-line response limits. The implementation of these requirements results in software complexities and inefficiencies in the following way. Pointer mechanisms for mapping structures and providing fast access paths have to be implemented by software and data. These pointers are extra data requiring extensive overhead in storage, access time, and maintenance.

Recent approaches to hardware support for nonnumeric processing

The limitations of conventional processors prompted the design of unorthodox architectures which could distribute processing by using parallel hardware configurations. The early trends were to use associative search hardware as subsystems within the central processor and the operations of these subsystems were closely associated with the operations of the central processor.[1] These systems, however, were restricted to small data bases because of their high cost. It is also evident that, with the present technology and the foreseeable future, it will not be feasible to build large scale pure associative memories as required by DBMS.

The desirable features sought in DBMS environment are:

(a) A large capacity and modular storage medium with low cost per bit,
(b) Ability to directly map logical data structures into physical data structures without using auxiliary structures,
(c) Variable length data formats,
(d) Fast retrieval and update of sets of data with respect to the requirements of on-line concurrent environments,
(e) Context (Boolean combinations of content) search operations assisted by total associativity,
(f) Complete instruction sets.

The solution to the cost limitations of associative memories in view of the desirable features of the DBMS environment was the introduction of the idea of using distributed logic in inexpensive large capacity circulating memory devices. Slotnick was the first to propose an associative file processor using a logic head per track device.[2] Healy and Parhami studied possible architectures which combined logic with a rotating bulk memory to achieve string and template matches in a context addressed manner.[3,4] Minsky proposed a scheme which involved an arrangement of the key items and data items grouped separately using the cylinder concept of a disk memory.[5] Parker partially designed a device that combined each head of a fixed head rotating memory with a single IC logic chip.[6] The studies mentioned thus far achieve only partial associativity and most of the fundamental DBMS operations had to be accomplished by the outside processor. Su, Copeland, and Lipovski were the first to study the design of a cellular processor on a rotating device to support the general data structures and requirements of DBMS.[7,8,9]

The overall design of the data structure, instruction set, and hardware architecture for RAP is presented. This design has been specified to the gate level. A discussion of cost and space estimates is presented in the conclusion. Details of this design can be found in a technical report.[10]

RAP ARCHITECTURE

Basic organization

An overall configuration of an operational RAP environment is given in Figure 1. RAP is an autonomous processor which communicates with an outside general purpose computer (GPC) only to receive its data base contents, to receive its compiled programs, and to send back the results of a user's requests.

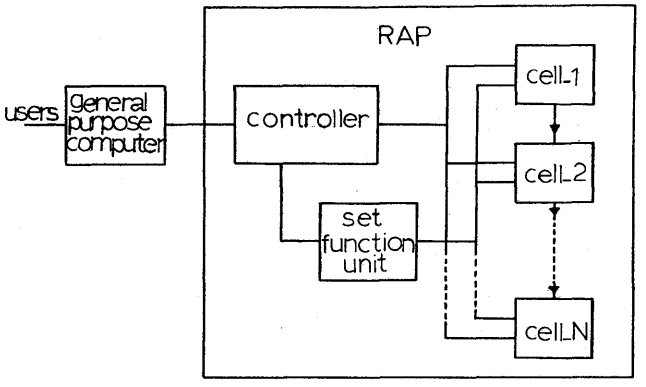

Figure 1—Overview of RAP architecture

The design is composed of a controller, an arithmetic set function unit, and a parallel organization of cells. A cell consists of a memory component and a logic component. The memory unit is one track of a rotating device such as a disk, drum, circular shift register, etc. The logic component is a microprocessor which acts as a "search machine" on data, directs data manipulation, and performs limited numeric computations required by data base processing. The set function unit is used to combine cell results to obtain a value computed over the total memory contents. The controller is responsible for overall coordination and sends control sequences to the cells, controls the set function unit, and executes decision commands and other RAP primitives that can be accomplished directly in itself.

The logical RAP data structure is stored directly so that no transformations are required. Data base contents are manipulated directly on their storage by their processors. The search criteria are transmitted to all of the cells simultaneously. It is evaluated as the memory contents "pass by" and the qualified contents are immediately manipulated or read out. Because the entire memory is processed for each instruction, inverted lists and other search aids would not enhance processing speed. More important, their absence necessarily eliminates the overhead of their maintenance. Since RAP has a parallel organization and its memory is associative, the response time is usually independent of data base size or content but does depend on query complexity. A query can be made up of one or more assembler instructions. Most instructions are executed within one rotation of the entire RAP memory contents.

Since data base management operates mostly in a concurrent environment, means must be provided to release the device as soon as possible from one operation to start another. It cannot tolerate excessive delays or overhead. The overall design philosophy is based on this principle. There are several hardware provisions throughout the design to achieve this principle.

Cell organization

Each cell consists of a rotating memory, a buffer, an information search and manipulation unit (ISMU), and an arithmetic logic unit (ALU). The basic logic blocks are displayed in Figure 2. Each cell lies on a serial communication path and receives status signals from its predecessor and sends signals to its successor. The cell also has connections for I/O and signals that are exchanged with the controller and set function unit.

Rotating memory

Data is read or written via fixed heads—one set for each cell—while the memory rotates under these heads. It takes one revolution of the store for its contents to be read from one end to the other. This time is called the latency. The associated memory with each cell is called a track. RAP memory space is the sum of the individual cell tracks.

A rotating bulk memory with high track capacity should be selected to achieve low cost per bit of storage. However, there is an upper limit on bit density since there is a processor associated with each cell. Equivalently, we require a lower limit on bit time—the time between two consecutively stored bits. It cannot be lower than a value determined by the speed of the processor circuits because logic and data/signal transmission must be completed in the time elapsed between two bits. A lower limit of 100 nanoseconds has been achieved as the bit time supported by the cell logic if implemented with the current IC technology.

Buffer

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

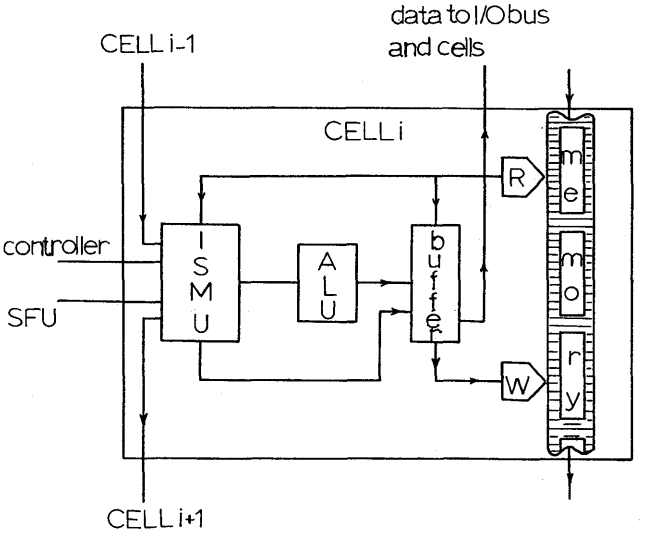

Figure 2—Overview of cell architecture

Data structure

A general data structure called RAP relations (similar to normalized relations as defined by Codd[11]) are stored with respect to a fixed track format. Details of the data structure and track format are given in the following sections. It suffices to say that a relation can be viewed as a "table" of data whose rows will make up the blocks of data stored on a track. Rows are called tuples because they represent a p-tuple of values. Both the ISMU and ALU circuits are designed to function on variable length data fields within the data blocks.

Contents of the memory are composed of blocks of data and/or garbage. The functions of storage allocation and garbage collection are accomplished directly in the hardware. Each cell uses the buffer to pack the data on the track toward the beginning of the track by "short circuiting" unwanted items. This accumulates the garbage at the end of the data. New data is first inserted at the garbage space of related tracks and, if more space is required, a new track is initialized and used to store the rest of the data. Garbage collection takes place in parallel with other operations without the need of user control or intervention.

Information search and manipulation unit

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

Arithmetic logic unit

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

Concurrency facility

The RAP concurrency facility employs a foreground-background principle implemented in hardware so that long programs can be run on preemptive basis.

Set function unit

The set function unit (SFU) provides the logic to calculate the set functions count, sum, maximum, minimum, and average over the entire cell memory. Intermediate results are stored in individual ALU registers of the cells. Arithmetic is accomplished through a serial adder and a serial divider which work on the intermediate results as they are collected from the cells. Also provided are counters for control sequencing as well as data gathering from the cell units and for transmitting the SFU result to the controller.

Controller

The controller is responsible for the overall coordination of the cell processors. It loads the search criteria units of ISMUs, energizes opcode and mode lines, senses the end of an operation, and repeats the cycle for the next primitive. The micro-orders cause stack buffer contents associated with instructions to be popped off and bussed onto cell lines during data block gap intervals. The controller also executes certain primitives that can be accomplished directly in itself by making use of the contents of registers whose values have been stored from previous operations.

Other operations, such as data transfers to and from the GPC, parity bit generation, serial-parallel conversion, and error checking procedures, which are common routines that exist in present devices, will be incorporated into the final design.

Role of the general purpose computer

A GPC to be chosen for interfacing with the RAP device should provide the following main functions:

(a) Support a data communication environment for users with proper I/O facilities,
(b) Compile user queries expressed in a high level query language into RAP primitives,
(c) Transfer compiled RAP primitives to the RAP controller with associated data used as operands in the format required by the RAP controller,
(d) Transfer data to cell storage for data base creation and expansion,
(e) Support a concurrent processing environment and thereby administer scheduling of RAP program entry queues,
(f) Control data base security and integrity,
(g) Maintain relation names, domain names, and data value encoding tables.

RAP DATA STRUCTURE

The data structure chosen for RAP is a modified version of the relational model introduced by Codd.[11] This model has been proven to be general enough to build other known useful data structures and, more important, present the same information conveyed as other data structures in a way that achieves a high degree of data independence.

RAP relational structure

We can view a normalized relation as a table of data about a set of similar entities. The table heading is the relation name, the column headings are the entity's attributes which are called the domain names, and a row represents a p-tuple, or simply a tuple, of values—one for each domain—which describes an entity. A relation con-

tains a varying number of tuples and a relational data base contains several interrelated relations through common domains.

A candidate key of a relation is a minimal combination of domains whose elements (values) uniquely identify every tuple in that relation. A primary key is one of the candidate keys with which the unique tuple identification is based for implementation.

A RAP relation is a normalized relation of the type described above except that duplicate tuples are allowed. One restriction is that the degree cannot be higher than a number pmax depending on the size of the values of a domain. This limitation is imposed by hardware considerations. It would suffice to state here that due to variable word length representation of RAP domains, the range for a RAP relation's maximum degree is $29 \leq pmax \leq 101$. We expect this to be sufficiently high enough for most applications. If this is not the case, then two or more relations can be made from the larger one. This is accomplished by partitioning the domains and repeating the primary key with each subrelation.

Track format

Since RAP eliminates intermediate mapping structures, a method of representing the data structure directly on the storage had to be found. Due to the linear nature of a memory track, a direct mapping of data structure would require it to be linearized. The relational structure lends itself to linear form easily as shown in Figure 3. The tuples of the relation, which are themselves linear representations of domain values, can be stored one after the other on a track. This structure is similar to a file on a conventional disk. The first two data blocks contain relation and domain names respectively and act as "header" blocks. Each succeeding block contains the concatenated value items in a tuple. The order of domain names determines the order of the values in each tuple. If a relation has too many tuples to be stored on one track, then several cell tracks are used. The hardware requires the relation and domain names to be repeated once on each track of a relation and that no cell can have tuples from more than one relation.

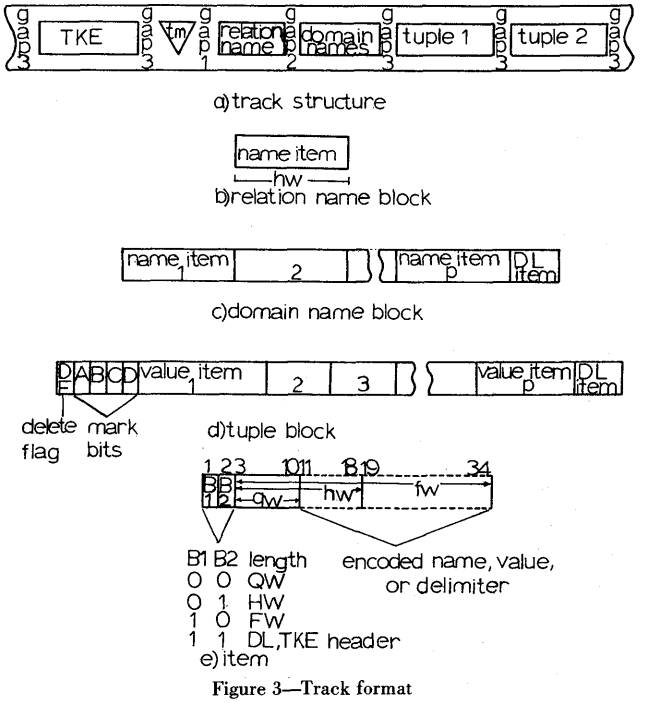

Figure 3—Track format

All names, values, and delimiters are items of encoded bit patterns. The relation name block is made up of one fixed length item. Domain and tuple blocks can have a variable number of items in different relations but not within the same relation. The end of these blocks is indicated by a delimiter item (DL). Blocks are separated by fixed length interrecord gaps. The beginning of a track is indicated by a marker which is detected electronically. This marker also implies the physical end of a track. The logical end of a track is indicated by a tuple block which carries a delimiting "track end" (TKE) item stored in the first value item's position. Items that make up domain and value blocks can also be variable length. However, these lengths must either be 32, 16, or 8 bit encodings. Each item is preceded by a two bit code to indicate the item's length. The fourth code is used to specify DL or TKE items which are distinguished from each other by the following bit. There is an upper bound on the length of a RAP relation tuple which is determined by the length of the cell buffer (1024 bits).

Tracks for a relation are allocated one track at a time as needed. Relation tracks are not required to follow sequential or contiguous track addresses since each relation track is identified separately. This results in different relation tracks intermixed with each other. By making each track a separate entity by itself and spreading the relations across noncontiguous cells, output efficiency is greatly increased. This is because several contiguous tracks are serviced as one channel group. Output occurs from one qualified cell in each channel group and outputs from these groups are interleaved to achieve a piped transmission.

There are five control bit positions at the beginning of each tuple. The first bit is the delete flag (DF). If this bit is on, it indicates that tuple is deleted and garbage collection hardware can "erase" the tuple. Garbage collection on each track is handled dynamically. The data is packed toward the beginning of a track. In case of insertions, all of the relation tracks are examined and their available garbage tuples are filled with incoming tuples. If still more insertions are to be made, a new cell and its relation track can be allocated automatically.

The A, B, C, and D are mark positions composed of one bit each. If a tuple is T-marked, T being any combination of these 4 bits, the corresponding mark bits are turned on. Likewise, a tuple is said to be T-unmarked if the T combination of bits are turned off. There are several instructions for marking tuples and/or using the markings as extra data values for qualifying sets of tuples. These

bits allow the results of one instruction to be used by another. This greatly extends the associative capability of RAP.

A fixed length gap is required between every two blocks. The lengths of these gaps are proportional to the amount and speed of logic required between block operations.

Preformatting of a track involves writing of all the control information of the block types on a track and separating the blocks by gaps. Relation and domain blocks must be filled in by their names. Tuple blocks are written until the physical end of the track. Their DL identifiers and the length codes in the first two bit positions of each item must also be filled in. The data portion of value items does not contain any information at that time. The logical track end is indicated by writing the identifier TKE in the first value item position of the very first tuple block. The first value item position of each tuple block starts at the sixth bit position leaving room for the delete flag and mark bit positions. Care is taken to make all of these five bit positions zero while preformatting each tuple block.

RAP INSTRUCTION SET

Basic relational operations

The RAP instruction set is designed to construct the basic operations necessary to support relational data bases and, at the same time, to be feasible for hardware implementation.

These basic operations are:

(a) Selection,
(b) Implicit join,
(c) Set operations,
(d) Projection,
(e) Free variables,
(f) Arithmetic set functions,
(g) Simple arithmetic update.

Selection is performed by applying a Boolean search predicate to each tuple of a relation and "marking" or reading the tuples satisfying the predicate. The implicit join operation allows values retrieved from one relation to be used as the retrieval criterion on another relation. The association is made through domains common to both relations. The set operations of union, intersection, complement, and difference can be done on the selected subsets of a relation. Projection is the act of selecting a subset of domains to be retrieved and eliminating duplicate values after a possible selection has occurred. A "free variable" is the term given to an implementation of the following capability.[12] It involves the selection of tuples based on the values of domains which occur in other tuples of the same relation. RAP programs which accomplish projection and free variables require explicit iteration.

Structure and operation of the instructions

Many RAP primitives reflect in their structure the basic characteristics of a DBMS query. These primitives specify an *operation* to be performed, a data *specification* section indicating what data is to be operated upon, and a *qualification* section specifying what conditions must be met before the operation can be fulfilled. The general format of these primitives is:

⟨LABEL⟩⟨OPCODE⟩[⟨SPECIFICATION⟩:
⟨QUALIFICATION⟩]

The label is an optional symbolic address of an instruction. The opcode specifies the operation. A specification has the format:

RN(DN1, DN2, . . ., DNK)

where RN is a relation name and DN1, DN2, . . ., DNK is an optional domain list. There is a hardware limit k on the number of K domains that can be included for any specification. A qualification is a Boolean expression of conditions on the values of the domains of a relation and on the mark bit values of the tuples in which the domain values are stored. It can take one of the following forms:

(a) Null (a qualification which does not specify any condition), every tuple of the relation is considered to satisfy the qualification,
(b) $Q1 \wedge Q2 \wedge \cdots \wedge QK$, denoting conjunction,
(c) $Q1 \vee Q2 \vee \cdots \vee QK$, denoting disjunction.

where K must be less than or equal to k—the hardware limit of the number of parallel comparators—and Qi is one of the following:

(1) ⟨RN⟩·⟨DN⟩ ⟨COMPARATOR⟩ ⟨OPERAND⟩,
—where DN is a domain name in relation RN,
—COMPARATOR is one of $=, \neq, <, \leq, >, \geq$,
—OPERAND is one of:

- ⟨REG⟩ (a register),
- integer,
- literal (characters bounded by quotation marks),
- a GPC program variable name (enclosed in parentheses),

(2) RN.MKED(T),
(3) RN.UNMKED(T),
where T is replaced by one of the following mark bit combinations:

A, B, C, D, AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD, ABCD.

Permutations are considered to be identical,
(4) CELL (I), where I is the integer identification of a cell.

Consider the following example. Assume a ternary RAP relation EMPLOYEEwith domains NAME, NCHILDREN, and SALARY and further assume that one of its tuples is stored in track format with the values shown below.

(a) DF=0
A=1
B=0
C=1
D=0
(b) NAME=CLARK
(c) NCHILDREN=3
(d) SALARY=9500

Some of the qualifications that this specific tuple satisfies are:

(a) NULL
(b) EMPLOYEE.MKED(A)
(c) EMPLOYEE.UNMKED(BD)
(d) (EMPLOYEE.NAME='CLARK') $\wedge$
(EMPLOYEE.MKED (AC))
(e) (EMPLOYEE.NCHILDREN $\neq$ 5) $\vee$
(EMPLOYEE. SALARY $>$ 5000) $\vee$
(EMPLOYEE.MKED(A))

At this stage, even without seeing the details of the instructions, it is possible to show how set operations can be implemented by using marking qualification logic. Assume that at one instance tuples of a relation are A and B-marked by criteria-1 and criteria-2 respectively. We denote the subset of tuples A-marked by criteria-1 by "SETA" and those B-marked by criteria-2 by "SETB". The following qualification statements can be written for the set operations:

(a) SETA$\cap$SETB (intersection) qualification:
RN.MKED(AB)
(b) SETA$\cup$SETB (union) qualification:
(RN.MKED(A))$\vee$(RN.MKED(B))
(c) SETA-SETB (difference) qualification:
(RN.MKED(A))$\wedge$(RN.UNMKED(B))
(d) $\neg$ SETA (complement) qualification:
RN.UNMKED(A)

Each command is formulated as an assembler language instruction for the RAP machine. One assembler instruction is implemented as one controller micro-code sequence which activates the hardwired logic in the cell circuits. The opcodes for each instruction will be given followed by a brief explanation. Instruction timings are given in the following section.

Retrieval commands

These commands are used in the process of locating the qualified rows of relations, and, if required, outputting them to the GPC. The commands included in this group are:

(a) MARK(⟨T⟩): Indicates the tuples of a relation which satisfy a Boolean condition on its values by setting "marking" bits,
(b) RESET(⟨T⟩): Resets marks,
(c) READ: Transfers fields of data from qualified and/or marked tuples,
(d) READ_REG: Transfers the contents of registers from the controller,
(e) CROSS_MARK(⟨T⟩): Marks a second relation based on the values of a previously marked relation (i.e., a hardware implementation of the implicit join operation),
(f) CRS_COND_MARK(⟨T⟩[⟨T^1⟩]):
A variation of CROSS_MARK; used for cross marking several relations into a single relation,
(g) GET_FIRST_MARK(⟨T⟩): Used for queries which involve the selection of tuples based on values which occur in other tuples of the same relation (e.g., used within a program implementation of the free variable and projection operations). It finds the first marked tuple of a relation, resets it, and takes the value of a domain from that tuple and marks other tuples in the same relation whose same or other compatible domain satisfy a comparison on that value. It can also store value items of a tuple into the controller registers to be used as arguments in subsequent operations.
(h) GET_FIRST: Places the specified values of the first marked tuple found into controller registers and resets the mark,
(i) SAVE: Stores values from a single tuple into controller registers.

Update commands

These commands provide value replacement by direct insertion or after arithmetic operations. The updated values are written back immediately so that a separate rewrite command is not required. The commands in this group are:

(a) REPLACE(⟨T⟩): Replaces the value of a domain of a set of qualified tuples with a new value,
(b) ADD(⟨T⟩): Adds a constant, contents of a controller register, or contents of another numeric domain to a domain of a set of qualified tuples,
(c) SUB(⟨T⟩): etc.,
(d) MUL(⟨T⟩): etc.,
(e) DIV(⟨T⟩): etc..

The optional marking capability of these instructions is used for recovery from device or software crashes. As each update takes place, the combinations of T-mark bits are set. If a crash occurs, the operations can be resumed on tuples with T-markings still off.

Set function commands

These commands compute functions over qualified sets of data. Similar to update commands the memory contents are evaluated "in place" so that transportation of large amounts of data is eliminated for simple computations. The commands included in this group are:

(a) SUM: Sums the values of a set of qualified tuples,
(b) COUNT: etc.,
(c) AVERAGE: etc.,
(d) MAX: etc.,
(e) MIN: etc..

Insertion and deletion commands

The commands are:

(a) INSERT: Inserts a new tuple into a relation,
(b) DELETE: Deletes qualified tuples of a relation,
(c) DROP_DOMAIN: Deletes a domain from a relation.

Data base creation and destruction commands

The commands are:

(a) CREATE: Preformats a cell so that it may be used to store and manipulate tuples of a relation,
(b) DESTROY: Removes all formats and data from a single cell or all cells containing data from a specified relation.

Decision and transfer commands

As in any programming language, certain instructions are required for testing system indicators, providing conditional and unconditional transfers within a program, and indicating termination. The following instructions provide these functions:

(a) TEST: Sets the contents of a status register based on the contents of markings,
(b) BC: Transfers control to another instruction if a condition is met,
(c) EOQ: Signals the end of a query.

SUMMARY OF INSTRUCTION TIMINGS

Table I gives the time required for the execution of each of the RAP instructions.

Table I—Instruction Timing

TYPE OF OPERATION	INSTRUCTION	EXECUTION TIME IN NUMBER OF REVOLUTIONS UNLESS OTHERWISE SPECIFIED
Retrieval commands	MARK	1
	RESET	1
	READ	Minimum=1/2 on average, Maximum=Number of qualified tracks occupied by the relation
	READTREG	Negligible
	CROSS_MARK	Depends on data base and k (Reference 10)
	CRS_COND_MARK	Same as CROSS-_MARK +1
	GETTFIRST_MARK	1 +Fraction
	GETTFIRST	1
	SAVE	1
Update commands	ADD	1
	SUB	1
	MUL	1
	DIV	1
	REPLACE	1
Set function commands	COUNT	1 +Fraction
	MAX	1 +Fraction
	MIN	1 +Fraction
	SUM	1 +Fraction
	AVERAGE	2 +Fraction
Insertion and deletion commands	DELETE	1
	INSERT	Minimum = 1, Maximum = 2
	DROPTDOMAIN	Maximum number of relation tuples per track
Data base creation and destruction commands	DESTROY	Fraction
	CREATE	1
Decision and transfer commands	TEST	Negligible
	BC	Negligible
	EOQ	Negligible

SELECTED SAMPLE QUERIES

Some sample queries will be programmed with the RAP instruction set. The samples are similar to those described in the literature for languages SQUARE,[12] SEQUEL,[13] and ALPHA[14] which were proposed as high level query languages for relational data bases. These languages have been proven to be complete in the sense that they are at least as powerful with respect to query formulation as the relational calculus. The RAP instruction set includes all the capabilities of these languages. All of the queries used as examples in the above languages have been programmed in RAP.

The data base used for the following examples consists of the following relations:

EMP(NAME, DEPT, MGR, SAL)
LOC(DEPT, FLOOR)
SALES(DEPT, ITEM, VOL)

The employee relation has a tuple for every employee giving his name, department, manager's name, and salary.

The location relation LOC gives the floor on which each department is located, that is, it has a tuple for each department floor pair. The SALES relation has a tuple indicating the yearly volume sold for an item in each department.

Q.1 Find the employee whose salary is greater than that of any employee in the SHOE department.

The RAP program is:

(a) MAX [EMP(SAL):EMP.DEPT = 'SHOE'],
(b) MARK(A) [EMP:EMP.SAL ⟩ (REGF_1)],
(c) EOQ

Q.2 List the names and managers of employees in the SHOE department with salaries greater than 10,000.

The RAP program is:

(a) READ [EMP(NAME,MGR):(EMP.DEPT = 'SHOE') (EMP.SAL ⟩ 10000)] [WORKAREA],
(b) EOQ

Q.3 Move the location of the TOY department to the second floor.

The RAP program is:

(a) REPLACE
[LOC(FLOOR):LOC.DEPT = 'TOY'] [2],
(b) EOQ

Q.4 Find the items sold by departments on the second floor.

The RAP program is:

(a) MARK(A) [LOC:LOC.FLOOR = 2] Mark departments on floor 2,
(b) CROSS_MARK(C)
[SALES:SALES.DEPT = LOC.DEPT:
LOC.MKED(A)]
Cross mark into SALES tuples the matching departments in the marked LOC tuples
(c) READ [SALES(ITEM):SALES.MKED(C)]
[WORK-AREA]
Read items sold by departments on the second floor
or
(b^1) CROSS_MARK(BC [SALES:SALES.DEPT
=LOC.DEPT: LOC.MKED(A)]
(c^1) (1) L1 GET_FIRST_MARK(A) [SALES:SALES.
ITEM= SALES.ITEM:SALES.MKED(b)]
(2) RESET(ABC) [SALES:SALES.MKED
(ABC)]
(3) TEST B-RAIL
(4) BC L1,RAILTSTAT(B)
(1) through (4) is a projection
operation on the answer found in b^1
(5) READ [SALES(ITEM):SALES.MKED(C)]
[WORKAREA]
read projected results that are
C-marked
(d) RESET(C) [SALES]
(e) EOQ

TABLE II—Cost and space estimates for a single cell

	CHIP COUNT/TOTAL COST (INCLUDING MULTIWIRE ™WIRING AND CIRCUIT BOARDS)		
Device complexity	k=1	k=5	k=5 & Concurrency
Catalogued SSI+MSI components	300/$425	399/$575	458/$657
All MSI (some to be made specially)	70/E$156	101/E$224	114/E$253
MSI+LSI (some MSI and all LSI to be made specially)	28/E$124	40/E$180	42/E$190

COST AND SPACE ESTIMATES

A preliminary logic design of the cell hardware has been completed to the gate level. This design has provided estimates for the cost and space requirements of the RAP hardware. The number of parallel comparator units k must be considered in terms of software implications and hardware costs. At this stage an intuitive value of 5 appears to satisfy a wide range of software applications. Table-2 presents the space and cost figures. E$ denotes estimated costs.

The relationship between the k values is linear. The R/W buffer is an existing LSI component. All components other than the LSI are constructed from T^2L some of which are Schottky and high speed types. The power dissipation for an implementation at the MSI scale would be 17, 25, and 29 watts per cell for k=1, k=5, and k=5 & concurrency respectively and a lower dissipation will result with LSI implementation.

The costs in the table are given for the full capabilities of the system, that is, all of the instructions are included. The first row gives the catalogued small quantity figures which can be used for the construction of an initial prototype. The controller circuitry dealing with the RAP functions is about the size of one cell. This excludes the read only and other buffer and stack memories as well as the circuits in the controller I/O function unit.

A review of current industrial technology indicates that disk memories could contain a track capacity of $0.5*10^6$ bits with a latency less than 50 milliseconds. Two hundred cells with this track capacity can accommodate a large data base on the order of 10^8 bits of compressed data. For very large data bases RAP can be used as an intelligent virtual memory to be backed up by conventional large bulk memories.

ACKNOWLEDGMENT

This research was supported in part by the National Research Council of Canada.

REFERENCES

1. Dugan, J. A., R. J. Green, J. Minker, "A Study of the Utility of Associative Memory Processors," *Proceedings of the ACM National Conference*, 1966.

2. Slotnick, D. L., "Logic Per Track Devices," *Advances in Computers*, Academic press, 1970.
3. Healy, L. D., K. L. Doty, G. J. Lipovski, "The Architecture of a Content Addressed Segment Sequential Storage," *Proceedings of FJCC*, 1972.
4. Parhami, B., "A Highly Parallel Computer System for Information Retrieval," *Proceedings of FJCC*, 1972.
5. Minsky, N., "Rotating Storage Devices as Partially Associative Memories," *Proceedings of the ACM SIGFIDET Workshop on Data Description*, Access, and Control, 1972.
6. Parker, J. L., "A Logic Per Track Retrieval System," *IFIP Congress*, 1971.
7. Su, S. Y. W., G. P. Copeland, G. J. Lipovski, "Retrieval Operations and Data Representations in a Context Addressed Disk System," *Proceedings of ACM Programming Languages and Information Retrieval Interface Meeting*, 1973.
8. Copeland, G. P., G. J. Lipovski, S. Y. W. Su, "The Architecture of CASSM: A Cellular System for Non-numeric Processing," *First Annual Symposium on Computer Architecture*, 1973.
9. Copeland, G. P., S. Y. W. Su, "A High Level Data Sublanguage for Context Addressed Segment Sequential Memory," *Proceedings of the ACM SIGFIDET Workshop on Data Description, Access, and Control*, 1974.
10. Ozkarahan, E. A., S. A. Schuster, K. C. Smith, *A Data Base Processor*, Computer Systems Research Group TR-43, University of Toronto, September 1974.
11. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *CACM*, 13, 6, 1970.
12. Boyce, R. F., D. D. Chamberlin, W. F. King III, M. M. Hammer, *Specifying Queries as Relational Expressions: SQUARE*, IBM Technical Report RJ 1291, IBM Research Laboratory, San Jose, California, October 1973.
13. Chamberlin, D. D., R. F. Boyce, "SEQUEL: A Structured English Query Language," *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, May 1974.
14. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, 1971.