TRENDS IN NON-SOFTWARE SUPPORT FOR INPUT-OUTPUT FUNCTIONS

Ken J. McDonell
Department of Computing Science,
University of Alberta,
Edmonton, Alberta T6G 2H1, CANADA.

Abstract

Input-output subsystem architectures have evolved over the past 20-odd years to the point where two divergent approaches have found acceptance in current computer systems; the 'IBM channel' is the archetype of the lower level alternative, while the functionally more complex techniques involve a wide spectrum of distributed processor architectures supporting database and/or storage management functions independently with respect to the central processor. The paper traces the historical development of support (outside central processor based software) for input-output functions and concludes with a preliminary comparison of the relative merits of the software interfaces provided by the alternative input-output subsystem architectures.

1 Introduction

A machine which supports an integrated database management system typically operates in an execution environment in which non-numeric operations are of primary importance. These non-numeric operations fall into two major classes; character oriented data manipulation within the central processor and input-output operations. This paper concentrates upon techniques, other than central processor resident software, which have been used to support the latter class of operations.

The presentation traces the historical evolution of firmware techniques and hardware architectures which have been incorporated into the input-output subsystems of computers destined to operate principally in a non-numeric mode. The impact of these changes upon the implementation of the input-output related data management functions is also discussed.

Within current computer systems, there are two commonly accepted input-output subsystem architectures, which differ significantly with respect to the distribution of processing capabilities between the central processor and the input-output subsystem. The later sections of the paper present the relative advantages and disadvantages of the software interfaces supported by these alternative architectures.

2 Some Miscellaneous Hardware/Firmware Techniques

Various techniques featuring ad hoc hardware or firmware assistance for input-output operations have been suggested or implemented. The following list gives some indication of the range of add-on options designed for increased operational efficiency or improved security.

* Autonomous data compression and expansion to reduce the stored data volume and the elapsed input-output time (Tao, 1974).
* Data encryption and password checking at the device level (Sindelar and Hoffman, 1974).
* Enforcement of memory protection boundaries during input-output transfers (Honeywell, 1975).

By and large, techniques of this nature may be incorporated within any of the input-output subsystem architectures and software interfaces described in the following sections.

3 Input-Output Subsystem Architectures

3.1 Early Configurations

Initial input-output subsystems provided simple, device dependent interfaces to the central processor. The datum transferred across the interface was determined by the characteristics of the peripheral device. For example, attached to an IBM 1401 (IBM, 1963), the 1402 card reader (punch) transferred a card image as 12 80-bit rows of information, while the 1403 line printer accepted zero to 100 characters in parallel. These early input-output subsystems were characterized by:

* little or no buffer storage outside main store,
* very limited control logic and data manipulation facilities outside the processor,
* no concurrency between multiple input-output transfers, even when associated with different devices, and
* complete dedication of main store and processor to input-output operations.

Under these early machine configurations virtually all the input-output processing was performed in software which was 'visible' to the programmer; e.g. record blocking and unblocking, character code conversion, buffer management, file label processing and format conversion.

3.2 Channel Controllers

By the mid 1960's the input-output architectures of most medium to large scale machines featured a radically different arrangement. Madnick and Donovan (1974, chapter 2) have attributed this change to the following facts:

* A dramatic increase in main store and processor speeds, relative to the speed of the peripheral devices. The consequent reduction in central processor throughput during input-output highlighted the inefficient use of the processor and main store.
* Substantial overlap of central processor operation and data transfers was technologically feasible using external logic which was specialized, simple, not too fast and cheap. These added logic modules provided a machine architecture which was economically attractive compared to the alternative designs based exclusively upon a general purpose central processor.

The resultant architectures feature five distinct functional units -- the central processor, main store and peripheral devices as in earlier machines, plus one or more channel controllers and a main store arbiter (refer to Figure 1). When coupled with an external interrupt mechanism and direct data paths between the channel controllers and main store (i.e. by-passing the central processor), this architecture permits parallel execution of central processor instructions with not just one, but many input-output transfers.

The central processor is typically interfaced to one or more channel controllers, which are in turn interfaced to the peripheral devices. By providing a single hardware interface for multiple heterogeneous devices, the channel controller masks many of the device idiosyncracies from the central processor. Physically, a channel controller may be constructed from two or more component modules (variously described elsewhere as input-output multiplexors, data channels and device controllers).

3.2.1 Channel Controller Functions

A process executing on the central processor initiates an input-output operation, or a sequence of operations by constructing a channel program from one or more channel commands. As an example, Figure 2 shows the semantics of a channel program to read one data record from an IBM 3330 series disk device (IBM, 1973), assuming the required hardware resources are available, and the record's physical address is known in advance.

Once a channel program has been constructed and placed in main store, the channel controller is called upon to fetch and interpret the channel commands, one-by-one. Within the channel controller local scheduling strategies must be implemented to ensure that conflicting channel programs may be interpreted in a sequence which guarantees their individual integrity.

Interpretation of a channel command involves the second area of responsibility for the channel controller, namely device control. These functions are often delegated to a separate device controller, capable of servicing multiple homogeneous devices on behalf of a channel program interpreter.

Co-ordination of transfers between main store and multiple active peripheral devices is also handled by the channel controller. Basically this involves multiplexing the data path(s) based upon the real-time urgency with which a given data transfer must be completed.

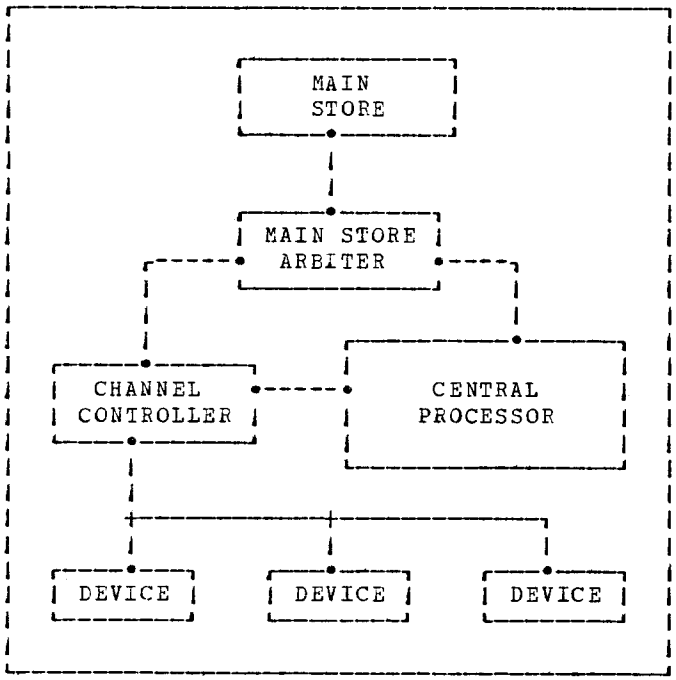

Figure 1 A Typical Channel Controller-Based Architecture

```
     SEEK                 cylinder/track addr
                          held in main store

LOOP:SEARCH ID EQUAL      ID (cyl/track/sector
                          addr) in main store

     BACK TO LOOP
        IF NOT EQUAL

     READ DATA            into main store
```

Figure 2 A Sample Channel Program to Read a Record

3.2.2 Main Store Accesses and Conflicts

The channel controller organization features multiple data paths to main store (at least one for each processor and each channel -- possibly more in the case of 'multi-port' and 'multi-bank' memories); in theory, requests for memory access could be generated on all data paths simultaneously (or at least within the same memory cycle).

Concurrent main store accesses are prevented from interfering by the main store arbiter. In the event that two requests for main store cannot be simultaneously honored, the arbiter permits one access and delays the other, based upon the priority of the requests. Usually the priority scheme ensures that transfers involving the fast devices are serviced before requests from the slower peripheral equipment or the central processor.

3.2.3 External Processing Capabilities

Besides freeing the central processor during input-output, channel controllers permit a small part of the processing associated with data transfers to be performed outside the central processor -- one of the most obvious examples being code conversion between the device and main store. However, some non-trivial features are also available; the 'File Scan' option on an IBM 2311 disk device/controller (IBM, 1969) supported the execution of simple character comparisons at the device (i.e. the SEARCH KEY AND DATA channel commands) -- this facility was subsequently enhanced and incorporated into the later 3330 series devices.

However, typical channel commands are so simple that their interpretation requires little external logic or processing capability -- to the extent that the central processor and a major part of the channel controllers for some machines (e.g. the IBM System/360 Model 50, (Flores, 1969, chapter 7)) are both implemented via microcode on the same physical processor. This simplicity is achieved at the cost of considerable central processor overhead associated with preprocessing and postprocessing the channel programs.

3.3 Input-Output Processors

In the multiprogramming environment of a general purpose computing facility there is considerable economic advantage associated with a system architecture which helps minimize the total time during which the central processor is either idle, interrupt processing, executing 'task swaps', or involved in mundane processing functions which could be handled by a smaller (cheaper) processor (e.g. a channel controller). But, if a channel controller is to achieve an even greater autonomy and capacity for parallelism with respect to the central processor, then clearly the channel programs must be able to initiate more complex data transfer sequences and to achieve data manipulation without central processor intervention. This implies the following general modifications to convert a channel controller to an input-output processor:

* Upgrade the command interpretation and control logic in the channel controller to the status of a bona fide (stored program) processor.
* Add considerable local storage for use as intermediate data and channel program buffers, and for holding the program(s) which interpret channel commands.
* Include device controllers with greater autonomy and a more sophisticated interface to the command interpreter.

An input-output processor accepts a requests for input-output operations from the central processor. In general, a channel controller would have to interpret many channel commands, spanning multiple channel programs, to achieve a result comparable to the interpretation of a single input-output operation. Note that the channel commands (or their equivalent) are constructed from the requested input-output operation within the input-output processor rather than at the central processor as is the case in a channel controller organization.

3.4 Distributed Support Processors

Input-output processors have been used to implement some of the functions provided by conventional operating systems. In a typical application, integral sections of the operating system (e.g. the input-output control system, the file system, or the resource manager) are off-loaded from the central processor to an input-output processor.

Examples of distributed support processors include:

* For a Control Data Cyber 70 or 6000 Series machine running under the Kronos operating system (CDC, 1976) the peripheral processors execute many of the monitor functions associated with physical resource management, job scheduling, input-output control, and system housekeeping.
* IBM's 3850 Mass Storage System (IBM, 1975) uses 'on-board' microprocessors to handle the migration of data within a 2-level storage hierarchy, and to maintain the necessary data set directories with minimal intervention by the central processor based operating system.
* The MICS system (Ohmori, Koike, Nezu and Suzuki, 1974) incorporates a 'File Processor Module' - currently a minicomputer - which allocates disk space to the computational processor(s), performs file management functions and handles all requests for disk transfers.

3.5 Backend Database Computers

A 'backend computer' is an extension of the input-output processor organization which was first used in the experimental data management system (XDMS) described by Canaday, Harrison, Ivie, Ryder and Wehr (1974). Within XDMS, a considerable portion of the Univac DMS 1100 database management system was transferred from the central processor to dedicated 'backend' (mini)computer. The backend processor independently executed commands expressed

at the level of the CODASYL Data Manipulation Language (CODASYL, 1971).

Heacox, Cosloy and Cohen (1975) have described some preliminary, but promising experiments with a 'Dedicated Data Management' processor architecture (modelled upon XDMS), while Lowenthal (1976) and Rosenthal (1977) have presented some of the rationale behind the backend approach currently being pursued by the MRI Systems Corporation.

The relative benefits and disadvantages of the backend organization fall outside the scope of the present discussion, however it will be assumed that this technique constitutes a viable alternative for database systems implementation.

3.6 Non-Numeric Processors and Storage Devices

Advances in hardware technology have eased the economic constraints on the development of viable architectures, specifically tailored for non-numeric processing (in particular database applications). Goals in this area include the reduction of input-output channel bandwidth requirements, alleviating the index maintenance overhead, increased parallel processing, response times which are independent of database size, and a closer match between the physical data storage structure and the user's conceptual information structure (Lipovski and Su, 1975; Ozkarahan, Schuster and Smith, 1975).

Since 1970, a great deal of investigation has been conducted into associative (or content addressable) secondary storage units (Copeland, Lipovski and Su, 1973; Coulouris, Evans and Mitchell, 1971; Mitchell, 1976; Ozkarahan et al, 1975; Lin, Smith and Smith, 1976). Most of these proposals and implementations employ parallel-serial searching to emulate content addressability on location addressable disk devices (i.e. a serial search over many tracks or areas simultaneously). Prototype devices in this class are quite powerful, being capable of many autonomous operations including, complex retrieval, update, deletion and garbage collection.

More recently, non-rotating storage devices has been used to build both content-addressable memory modules with asynchronous search capabilities (e.g. the RAP prototype and Honeywell's ECAM (Anderson and Kain, 1976)) and highly parallel search engines (e.g. the LEECH processor (McGregor, Thomson and Dawson, 1976)).

3.7 Database Processors

Perhaps the greatest potential for the use of external processing capabilities in a database environment lies with the development of backend 'database processors' based upon novel, non-numeric architectures (as opposed to off-loading software into a conventional minicomputer).

For example, a processor incorporating associative-type search modules, a tailored instruction repertoire, hardware aided database security and concurrent access control mechanisms, and firmware driven data structure translation could conceivably support a very efficient, integrated database management kernel similar to the one proposed by Lowenthal (1977). The availability of such a database processor would ease the development costs for new applications, whilst providing an acceptable throughput rate, a stable interface to the central processor, highly reliable operation and considerable capacity for local performance tuning and technological adaptation within the database processor itself.

4 Functional Attributes of the Input-Output Interface

The balance of this presentation is concerned with the choice of suitable input-output interface(s) to secondary storage, given the hardware scenario outlined earlier, the viability of special purpose (non-numeric) distributed processor architectures to support the input-output functions, and the apparent persistence of channel controller based architectures in current machines. The assumed hardware configuration is shown in Figure 3.

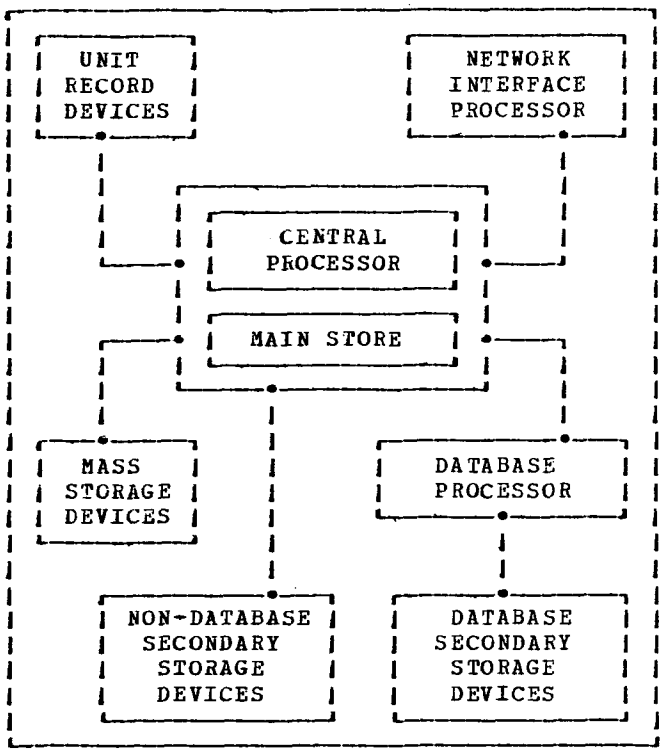

Figure 3 Architecture for a General Purpose Computing Facility

There have been many proposals for an interface between a program requesting an input-output operation and the input-output subsystem. For the purposes of the current presentation, interfaces involving 'stream' input-output devices will be ignored in favour of those involving secondary

storage devices because, (a) the main area of interest for this presentation has been defined as machines supporting database management systems (and by implication, databases resident on secondary storage), and (b) often the stream devices are actually spooled via secondary storage.

When viewed from a program executing on the central processor, the input-output subsystem provides a set of functional capabilities which may be selectively invoked; these capabilities define the input-output interface. Channel controllers support interfaces with a physical record orientation, while either a logical record or a logical file interface is provided by the input-output processor architectures. The essential differences between these two approaches are presented in Table 1.

All physical record interfaces are basically alike, however the logical interfaces vary significantly with respect to the complexity of the inter-record relationships maintained by the input-output processor. At the simplest level, the 'logical data interface' suggested by Casey (1973) uses an external processor to implement the mapping from record identifiers to physical addresses -- all program generated access requests are made in terms of a logical file name and a unique record identifier. At the other extreme are the interfaces to the special purpose database devices and processors, which provide operations specifically tailored for highly structured data models, for example:

* Relational data models and their associated non-procedural query languages; RARES (Lin et al, 1976) and RAP (Ozakarahan et al, 1975).
* CODASYL-like data manipulation languages for network data models; XDMS (Canaday et al, 1974) and DDM (Heacox et al, 1975).
* Set processing, based upon 'extended set theory' (Hardgrave, 1975).
* Flexible, query oriented data models for records with multiple secondary key fields; CASSM (Copeland et al, 1973), CAFS (Mitchell, 1976) and the 'database computer' (Baum, 1975).

5 Choosing an Input-Output Interface

For current computer systems, gross economic trends (Boehm, 1976) dictate that the choice of input-output interface should be primarily based upon software considerations, rather than relative hardware-costs -- demonstrable savings in software development and maintenance costs must predicate any general shift from the 'hardware status quo'. Consequently, the following discussion aims to present a software-oriented justification for the input-output interface design decisions.

5.1 How Many Interfaces?

One basic issue concerns the number of interfaces to be concurrently supported. On a preliminary basis, the implementation of a single input-output interface can be justified as follows:

PHYSICAL RECORD INTERFACE (channel controller)	LOGICAL RECORD INTERFACE (input-output processor)
Calling program must supply actual secondary storage address of the record to be transferred.	Calling program is unaware of blocks or the physical address of the requested record.
Main store buffer address must be explicitly supplied via a channel command.	Buffer location is implicit, or defined by symbolic reference.
The logical relationship between consecutive blocks of secondary storage is defined by the software executing on the central processor.	Any logical relationship between blocks of secondary storage is transparent to central processor based software.
Channel controller requires multiple main store accesses to retrieve channel commands and associated operands.	Input-output processor requires at most one main store access to retrieve the descriptor for an an input-output operation.
Semantic interpretation of the transferred data is performed entirely by software executing on the central processor.	Input-output processor is aware of the transferred data's semantics, from the input-output operation descriptor and from information stored within the input-output processor.

Table 1 A Comparison of the Functional Interfaces Supported by a Channel Controller and an Input-Output Processor

* There is no logical requirement for multiple interfaces. For example, fetching a 'page' of main store from backing store is logically equivalent to retrieving a record from an integrated database.
* The development and maintenance of input-output oriented software would be simplified, since all software would interact with one autonomous process charged with executing all secondary storage operations.
* The existence of the current heterogeneous software interfaces is a historical anomaly resulting from the development of functionally related, but physically disjoint, input-output support systems on top of a low level hardware interface (i.e. channel programs). For example, current computer systems typically support input-output routines for sequential devices, spooling, paging, the file system, 'stand alone' file access methods and a database management system. Routines shared between these subsystems have generally evolved in an ad hoc manner, compounded by the peculiarities and inadequacies of the precursor subsystems. This evolutionary approach results in the duplication of software development effort, unnecessary complexity within the input-output support modules, and poor utilization of physical resources (e.g. the spooling input-output routines may demand dedicated control of a complete device).

It must be stressed that adoption of a uniform interface has an effect which extends beyond the database management system, because all software modules which require access to secondary storage have to be considered. In the following two sections, both the physical record and the logical record interfaces are studied with respect to their ability to replace the current spectrum of heterogeneous software interfaces with a single, flexible and economic interface for all secondary storage input-output.

5.2 The Physical Record Interface

Obviously all operations could be carried out at the channel program level -- after all, this is the way most current systems are implemented -- however the resultant code duplication, device dependent procedures and program complexity renders this option increasingly unattractive.

Historically, low level device control and a physical record interface were provided in response to,

* a monoprogramming environment in which individual programs exploited their control over dedicated devices to reduce run time by overlapping input-output with processing, and
* machine architectures in which the central processor intervention was required to perform input-output related processing.

The introduction of multiprogramming systems has meant that an individual program no longer has (guaranteed) dedicated control over a device and the responsibility for achieving overlap of operations has been assumed by the operating system. In additon, reliance upon the central processor for input-output processing may be reduced by the adoption of distributed processor architectures.

Execution of a single database operation involves the use of multiple, non-trivial channel programs. Constructing complex channel programs in the central processor is inefficient, especially if the addresses of all operands and buffers must be translated from the virtual to the real address space (Buzen, 1975).

Despite the external processing capabilities of channel controllers, the bulk of the input-output processing required to implement 'data management level' operations (e.g. directory searching, logical record construction, and access via inter-record pointers) still has to be performed by software, executing on the central processor between channel programs. This situation is caused, in part, by the repertoire of channel commands for disk devices not being particularly well-suited to database operations; e.g. searching at the device is restricted to one key area in a fixed record format, while for short (e.g. index) records, device level searching may be less efficient than software searching through a block of records in main store (Buzen, 1975). In additon, many applications require multiple physical records to be transferred to main store before a single logical record or query response can be constructed -- this means an unnecessarily high bandwidth data path and more work for the main store arbiter.

A physical interface is unsuitable for the input-output modules which support even moderately complex functional capabilities -- e.g. a database processor or associative storage module -- since these units are designed for optimal performance when operating independently on a logical data structure at the schema or subschema level.

When attempting to show the unsuitability of a physical record interface as a homogeneous interface, it should be noted that the channel program mechanism may indeed be adequate, but the channel commands themselves should not specify operations on physical records.

5.3 The Logical Record Interface

The other alternative for a uniform interface is the logical record approach (as supported by most input-output and database processors). As an immediate consequence, all software would achieve a much larger degree of physical device independence than typically associated with the channel program interface.

Much of the searching function may be implemented outside the central processor, and transfers between main store and the input-output processor involve self contained logical records (e.g. a database record as defined by the subschema) or logical files (e.g. the tuples satisfying a query on a relational database). Therefore the bandwidth requirements and the degree of main store contention is reduced (compared to a channel controller interface capable of achieving the same logical record throughput).

Adoption of a logical record interface would enhance the modularity of the global system structure, allow improved resource availability and utilization, and help define the functional dependencies between the operating and database management systems (McDonell, 1977).

A logical interface would permit specification of operations whose functional complexity is well suited to the capabilities of a database processor or special search module.

While a logical interface appears to have significant benefits over its physical counterpart, not all logical record interfaces are the same. Chosing a logical interface capable of efficiently supporting all secondary storage access requests from application programs, the operating system and the database management system involves a set of problems and considerations too large to be included in the present discussions. However there are apparent advantages associated with the adoption of a logical record interface which has a procedural approach to record manipulation within a well defined structural data model, namely:

* The alternative interfaces designed for non-procedural access to tabular data structures (e.g. the query interface to a relational database) are highly desirable for a non-programmer's conceptualization of the data manipulation functions. But, they are less suitable for an interface to central processor resident software, since this interface is basically procedural and must be comprehended principally by programmers.
* Many of the non-database activities associated with secondary storage can be naturally implemented using a procedurally oriented logical record input-output interface; prime examples include spooling, paging and implementation of a general purpose user file system.

6 Conclusion

Input-output subsystem architectures have undergone significant evolutionary change, motivated by a desire for increased parallel operation, better central processor utilization and technological factors influencing the economics of processor construction. In terms of functional complexity, the trend culminates in the emergence of powerful database and input-output processors.

A logical record interface for secondary storage input-output appears to have considerable advantage over the alternative physical record approach. The logical record interface is convenient and efficient for communications with input-output processors, provides device independence for the central processor resident software and is simultaneously suitable for efficient implementation of a database management system and for use by all non-database procedures requiring access to secondary storage.

However, the following interface questions remain open for continued investigation:

* Should content addressable devices and database search engines be attached to an input-output processor, or directly to the central processor? Should these units have special interfaces, or are the standard interfaces adequate? The operational and functional relationship between the content addressable and location addressable devices remains unclear.
* Can a logical interface be constructed to provide the throughput and response required by high speed processes (e.g. paging)?
* How much of the database management system can be off-loaded to a database processor? This will have a critical effect upon the level of the logical interface.
* What will the impact of distributed databases be upon the logical input-output interface? Should the host's secondary storage interface accommodate the network interface protocol?

Acknowledgement

The author would like to thank the referees for their constructive comments on an earlier version of this paper.

References

Anderson, George A.; Kain, Richard Y. (1976): "A Content-Addressed Memory Designed for Data Base Applications", Proc. 1976 Internat. Conf. on Parallel Processing, Wayne State U., Michigan, August, IEEE Comp. Society, California, pp 191-195.

Baum, Richard Irwin (1975): "The Architectural Design of a Secure Data Base Management System", Report OSU-CISRC-TR-75-8, Computer & Information Sci. Research Center, Ohio State U., November, NTIS report AD A021 158.

Boehm, Barry W. (1976): "Software Engineering", IEEE Trans. on Computers, vol C-25, no 12, pp 1226-1241.

Buzen, Jeffrey P. (1975): "I/O Subsystem Architecture", Proc. IEEE, vol 63, no 6, pp 871-879.

Canaday, R.H.; Harrison, R.D.; Ivie, E.L.; Ryder, J.L.; Wehr, L.A. (1974): "A Back-end Computer for Data Base Management", Comm. ACM, vol 17, no 10, pp 575-582.

Casey, D.P. (1973): "Logical Data Interface", IBM Technical Disclosures Bulletin, vol 16, no 4, pp 1203-1207.

CDC (1976): "Cyber 70 Series Models 72, 73, 74 and 6000 Series Computer Systems: Kronos 2.1 Workshop Reference Manual", Publication No. 97404700, Control Data Corporation, April.

CODASYL (1971): Data Base Task Group Report, CODASYL DBTG, April, ACM, New York.

Copeland, George P.; Lipovski, G.J.; Su, Stanley Y.W. (1973): "The Architecture of CASSM: A Cellular System for Non-numeric Processing", Proc. 1st Annual Symposium on Comp. Architecture, co-sponsor IEEE Comp. Society, Florida, December, pp 121-128.

Coulouris, G.F.; Evans, J.M.; Mitchell, R.W. (1971): "A Hardware Aided Approach to Content-addressing in Data Bases", Hardware Software Firmware Trade-offs: Proc. IEEE Comp. Society Conf., Boston, Massachusetts, IEEE, New York, pp 19-20.

Flores, Ivan (1969): Computer Organization, Prentice-Hall Inc., Englewood Cliffs, New Jersey.

Hardgrave, W.T. (1975): "Set Processing in a Network Environment", ICASE Report No. 75-7, Universities Space Research Assoc., Hampton, Virginia, March, NTIS report N75 21035.

Heacox, H.C.; Cosloy, E.S.; Cohen, J.B. (1975): "An Experiment in Dedicated Data Management", Proc. International Conf. on Very Large Data Bases, Massachusetts, September, ACM, New York, pp 511-513.

Honeywell (1975): "Series 60 Level 66: Summary Description", File No. 1P01, Honeywell Information Systems Inc.

IBM (1963): "System Operation Reference Manual: IBM 1401/1460 Data Processing Systems", form A24-3067-0, IBM Corporation.

IBM (1969): "Introduction to System/360: Direct Access Storage Devices and Organization Methods", form GC20-1649-4, IBM Corporation.

IBM (1973): "Reference Manual for IBM 3830 Storage Control Model 1 and IBM 3330 Disk Storage", form GA-26-1592-3, IBM Corporation.

IBM (1975): "Introduction to the IBM 3850 Mass Storage System (MSS)", form GA32-0028-2, IBM Corporation, July.

Lin, Chyuan Shiun; Smith, Dianne C.P.; Smith, John Miles (1976): "The Design of a Rotating Associative Memory for Relational Database Applications", ACM Trans. on Database Systems, vol 1, no 1, pp 53-65.

Lipovski, G. Jack; Su, Stanley Y.W. (1975): "On Non-numeric Architecture", ACM SIGARCH Comp. Architecture News, vol 4, no 1, pp 14-29.

Lowenthal, Eugene I. (1976): "Backend Machines for Data Base Management: A Tutorial", Proc. 5th Texas Conf. on Computing Systems, U. Texas, Austin, October, IEEE Comp. Society, Long Beach, California, pp 21-25.

Lowenthal, Eugene I. (1977): "A General Purpose DBMS Kernel", presented at the 1977 USAFA Computer Related Information Systems Symposium (CRISYS), Colorado Springs, Colorado, January.

Madnick, Stuart E.; Donovan, John J. (1974): Operating Systems, McGraw-Hill Book Co., New York.

McDonell, Ken J. (1977): "User, Operating System, Database Management System: Global Structure and Functional Dependencies", presented at IEEE Comp. Society's Workshop on Operating and Data Base Management Systems, Northwestern Univ., Illinois, March.

McGregor, D.R.; Thomson, R.G.; Dawson, W.N. (1976): "High Performance Hardware for Database Systems", Proc. 2nd International Conf. on Very Large Data Bases, Brussels, Belgium, September, ACM, New York.

Mitchell, R.W. (1976): "Content Addressable File Store", presented at Online Conf. on Database Technology, London, England, April.

Ohmori, K.; Koike, N.; Nezu, K.; Suzuki, S. (1974): "MICS - A Multi-microprocessor System", Proc. IFIP Congress 1974, Stockholm, August, North-Holland, Amsterdam, pp 98-102.

Ozkarahan, E.A.; Schuster, S.A.; Smith, K.C. (1975): "RAP - An Associative Processor for Data Base Management", Proc. AFIPS National Comp. Conf., vol 44, Anaheim, California, May, AFIPS Press, Montvale, New Jersey, pp 379-387.

Rosenthal, Robert S. (1977): "An Evaluation of Backend Data Base Management Machines", presented at the 1977 USAFA Computer Related Information Systems Symposium (CRISYS), Colorado Springs, Colorado, January.

Sindelar, Frank; Hoffman, Lance J. (1974): "A Two Level Disk Protection System", Memo ERL-M452, College of Engineering, U. of California, Berkley, May, IEEE Comp. Society Repository R75-86.

Tao, W.Y.Y. (1974): "A Firmware Data Compression Unit", Report UIUCDCS-R-74-617, Dept. Computer Science, U. of Illinois, Urbana, January.

Tomlin, E.L. (1973): "Microprogrammed Disc Controllers", M.Sc. Thesis, Naval Postgraduate School, Monterey, California, December, NTIS report AD 789 812.