AD-A037 146 NAVAL ELECTRONICS LAB CENTER SAN DIEGO CALIF F/G 5/2
COMMAND CENTER INFORMATION SYSTEM (CCIS) FUNCTIONS AND CAPABILI--ETC(U)
NOV 76 D L SMALL, D O CHRISTY

UNCLASSIFIED NELC/TD-498 NL

| OF |
AD
A037146

END

DATE
FILMED
4-77

Technical Document 498

NELC / TD 498

COMMAND CENTER INFORMATION SYSTEM (CCIS)

Functions and Capabilities

DL Small
DO Christy

10 November 1976

Research and Development, March 1974 to November 1976

Prepared for:
Naval Electronic Systems Command
Command Control Division, Code 330
Washington, DC 20360

NAVAL ELECTRONICS LABORATORY CENTER
San Diego, California 92152

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
NELC Technical Document 498 (TD 498)

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*
COMMAND CENTER INFORMATION SYSTEM (CCIS)
Functions and Capabilities

5. TYPE OF REPORT & PERIOD COVERED
Research and Development
March 1974 to November 1976

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
DL Small
DO Christy

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Electronics Laboratory Center
San Diego, CA 92152

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
62721N; F21211; XF21211002/
XF21211001 (NELC N713)

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Electronic Systems Command (Code 330)
Washington, DC 20360

12. REPORT DATE
10 November 1976

13. NUMBER OF PAGES
14

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*

15. SECURITY CLASS. *(of this report)*
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*
Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Shipboard computers	Magnetic disks
Computer systems hardware	Data links
Computer memories	Relational data management
Serial access computer storage	Extensible query languages
Random access computer storage	

403 940

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The Command Center Information System (CCIS) provides a question-answering facility for relationally organized Navy data bases. The system consists of a query (user) processor, connected to a data processor by a 1200-baud communication line, a buffer memory, an interface processor, and bulk storage. Data-flow control between the interface processor and buffer memory is exercised by the data processor, the principal function of which is data manipulation. The buffer memory provides buffering between bulk storage (currently a disk memory) and the data processor.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

The processing organization is developed as follows: principal processing of the input user statements, which are in a form of extensible English, is performed by the query processor which provides the dialog with the user and parses the English statements. The query processor translates the parsed statements into an action sequence which, in turn, drives the data processor in its manipulation of relations and records or which updates the dictionary of functions, vocabulary words, and relations stored on the query processor's floppy disk.

The data processor has control over the disk-data management, stores and retrieves data as needed by the query processor, controls the interface processor, and loads the programs and data structures for both the user and interface processors. Special functions are available for relational data management such as mapping, mapping-composition, creating and deleting relations, creating, deleting, and replacing rows of a relation, projecting, prime-symbol mapping, and search relation, for memory management such as page and logical record management for buffer, disk, and local memory, and for scheduling and control. Special instructions are available to isolate the access to data (potentially useful for maintaining data security), for conditional jumps to a cell (to aid in implementing the scheduling monitor concept as available in Concurrent PASCAL), and for expanding and compressing data retrieved from and stored on the disk, respectively.

In operation, the query (user) processor looks up words in the user's dictionary by accessing a highly portable auxiliary memory, such as a floppy disk, through double hashing techniques. A word identifier is returned when the words are found. A speller looks up words in the dictionary which may have been misspelled by the user.

When a statement has been found in the input to the query processor, it loads, from the floppy disk, action descriptions for each of the key words found in the pruned parsed statement. A sequence of action statements is transmitted to the data processor where their execution provides a final result, such as one or more relations. The resulting record is transmitted to the query processor for format control, display control, and display. The query processor can also act as a text editor for the query statements.

The interface processor operates like a data-transformation unit. Only a simple syntax is processed by this processor and its principal functions are to recognize data structures of the retrieval system and to relate the incoming data with the appropriate records in the retrieval system. Finally, it writes the data onto the buffer memory for final disposition by the data processor.

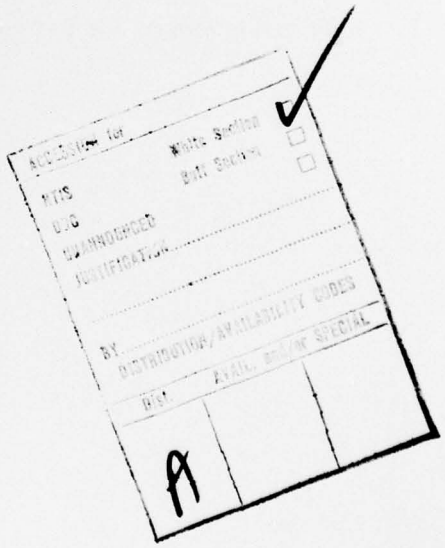
By architecturally partitioning the CCIS into three processors, each can be organized for its specialized interface (the user, bulk data storage access, and other computing systems). The interface and data processors and bulk storage must be physically colocated because of their high-speed communication requirement. Since the query processors communicate with the data processor at a lower data rate, they can be located at remote network sites without degrading system performance.

ACKNOWLEDGMENT

The authors thank Mr Henry Gok and Mr Jack Pigniolo of the Communications Processing Division, NELC Code 3200, and Drs Frederick Thompson and Bozena Thompson of the California Institute of Technology for their continuing contribution to the development of the Command Center Information System.

ADMINISTRATIVE INFORMATION

This work was performed by members of the Communications Processing Division for the Naval Electronic Systems Command, Code 330, under Program Element 62721N, Project F21211, Task Area XF21211002/XF21211001 (NELC N713). This document was approved for publication 10 November 1976.



CONTENTS

ILLUSTRATIONS

TABLE

INTRODUCTION

This document details the Command Center Information System (CCIS), a new architectural concept for Navy Information Storage and Retrieval (ISAR). The system consists of a query (user) processor, connected to a data processor by a 1200-baud communication line, a buffer memory, an interface processor, and bulk storage (fig 1).

The architecture design features a functional separation of translation of natural or formal language queries into data retrieval, update, and processing commands from the execution of these commands in a back-end data processor. In this manner, query translation and command execution are carried out by separate processors in a manner conceptually similar to the Bell Telephone Laboratories front-end, back-end processor system.[1]

Updating of rapidly changing data is performed in a third processor. Communication between processors or between processors and disk storage (except between the user processor and the other processors) is accomplished, first, by requesting data transfer via the controller of a four-port buffer memory and, then, by actually transferring data from disk to buffer and from buffer memory to processor (see fig 1).

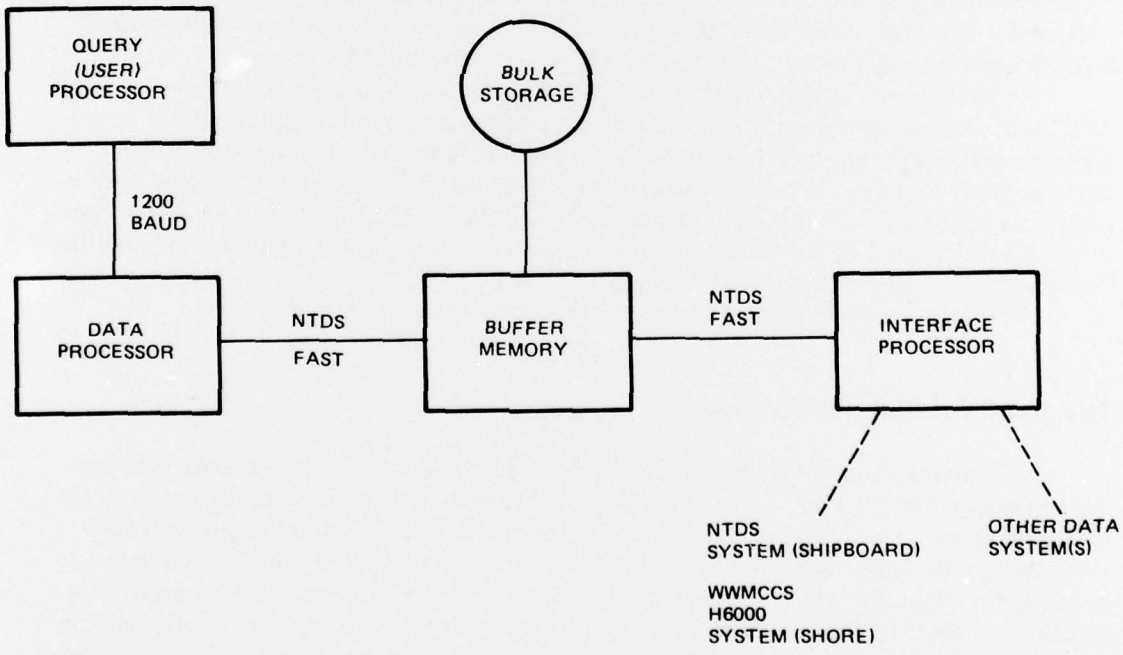


Figure 1. Command Center Information System block diagram.

1. Canady, RH, et al, "A Back End Computer for Data Base Management," Communications of the Association for Computing Machinery (CACM), October 1974

Implementation of the architecture depends upon the use of the latest microprocessor technology, query translation, and data retrieval. Update and processing command execution will use special microcomputers whose instruction repertoires are specifically designed to optimize execution of such commands. All buffer memory-to-processor communication is via high data-rate input and output channels.

INFORMATION STORAGE AND RETRIEVAL SYSTEM PROCESSOR FUNCTIONS

Early architecture implementations for this ISAR system will use a query and update language with limited syntax; however, language-defining facilities, a natural language parser, and a semantic interpreter will be implemented so that a language with a richer syntax and extension facilities similar to those implemented in Thompson's Rapidly Extensible Language System[2] can be added easily later. The latter will permit addition of new vocabulary elements and new data relationships by the system user (see the sample query session in NELC Technical Note 2782, Initial Command Center Information System Capability, 5 September 1974, by DL Small and DR Duke, for a potential Navy application).

Thompson's system also features optimization of relational data-base retrieval.[3] Thus, queries are optimized when they require a search for one or more specific elements in a class (set of elements with a similar characteristic, such as all ships), or when they require a search for the image of a class of elements in a large binary relation (a relationship which holds between two elements). The architecture will preserve this feature.

In addition, the architecture will support multiple user interactions with one or more data bases. A language specifically designed for writing syntax and semantic routines to support interpretive execution of user queries will be provided as well. This meta-language, based primarily upon relational primitives, some of the Rapidly Extensible Language System (REL) macros (such as the REL paging macros[4]), and the usual arithmetic and Boolean operators will be supported by software and a special microcomputer instruction repertoire designed for optimal execution of the meta-language.

The functional or logical structure of the system is shown in table 1.

THE QUERY (USER) PROCESSOR

The function of the query (user) processor (fig 2) is to provide the user with real-time translation of his English-like statements and queries into the control language of the data processor. The query processor provides the translation function in a manner such that the data processor does not have to execute this function in addition to the data-handling functions for which it is optimized. The translation process is a high-processing function, so that, by dedicating this function to the query processor, more queries can be handled. Several query processors can be attached to the data processor without noticeable degradation in response. In addition, the query processor provides buffering and editing functions for the user that do not contend with other users and with the data-updating function.

2. California Institute of Technology REL Report 3, REL – An Information System for a Dynamic Environment, by BH Dostert, December 1971
3. California Institute of Technology REL Report 4, Computer System Support for Data Analysis, by NR Greenfield, March 1972
4. California Institute of Technology REL Report 17, The REL Paging System, by FB Thompson, 1974

TABLE 1. INFORMATION STORAGE AND RETRIEVAL (ISAR) FUNCTIONS.

Query (user) processor

1. Editor module
2. Lexicon handler
3. Speller
4. Parsing
5. Semantic translator

Data processor

1. Scheduler
2. Executive
3. Relational data base management
4. Disk page management
5. Local and cache memory management
6. MIMC instructions

Interface processor

Auxiliary memory subsystem

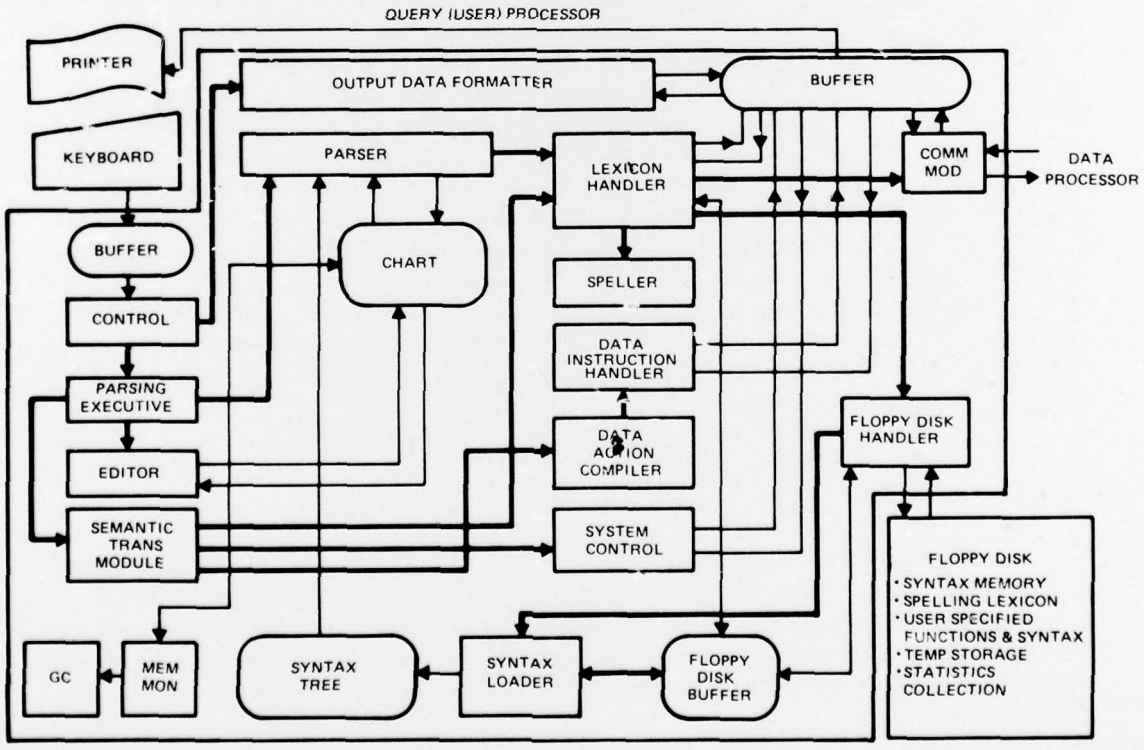


Figure 2. Query (user) processor block diagram.

The primary functions of the query processor are to provide editing to the user, to parse the English or formal language query, and to generate the appropriate instructions to the data processor for query response. A number of secondary functions, such as an executive which controls the order of activity (parsing executive), are required to accomplish these primary functions. This executive module calls an editor module which performs appropriate edit control and parsing. Once a sentence is parsed, it calls the translation module (semantic translator). In figure 2, these calls are denoted by double-lined arrows to the appropriate boxes.

The editor simply changes the parsing chart to correspond to the changes requested by the editing action. The chart stores the partially parsed statement, which the user has already entered into the translator, and provides all needed information for editing, continued parsing, and translation. A chart is built up for each new sentence entered into the translator. The parser takes the input from the parsing executive and builds on the chart by communicating with the lexicon module and with the syntax memory of the data processor in order to acquire new words entered by the user and new syntax rules for parsing. The lexicon handler uses a fast hash-code strategy for word lookups from the portable auxiliary memory (currently a floppy disk). The syntax handler gets new syntax rules from the syntax memory via the floppy disk buffer and the floppy disk handler.

The semantic translator takes a completed chart, prunes unnecessary paths from the chart, and formulates a course of action for the handling of data. It also determines whether the action is a meta action (a new definition), in which case it calls the syntax handler to see if a new syntax has been generated which must be added to the system. The semantic translator module calls system control if a system action is involved, such as formulating a new language or changing the effective data base for which subsequent statements are to be applied, and it also directs the user to his user preference history on the floppy disk. In the case where new words are being added to the system, the semantic translator module calls the lexicon handler to create a new word for the lexicon, calls the data action compiler for the reduced chart, and creates a sequence of commands for the data processor to carry out any desired file processing. The data action compiler calls the data-handling instruction module to communicate with the file handler of the data processor.

Each handler has a special protocol for carrying out the particular module's necessary action. Processing and protocol go hand-in-hand to reduce execution time of the statement. In order to limit the size of the query processor memory, excessive information from the data processor is minimized.

THE DATA PROCESSOR

The data processor is responsible for scheduling, fetching, updating, and analysis of data, the executive or semantic command interpretation (execution of commands as received from the translator), and storage-management processes. The latter consists of relational data-base maintenance, disk-page management, and local- and buffer-memory management. The hierarchy of functions for the data processor is shown in figure 3.

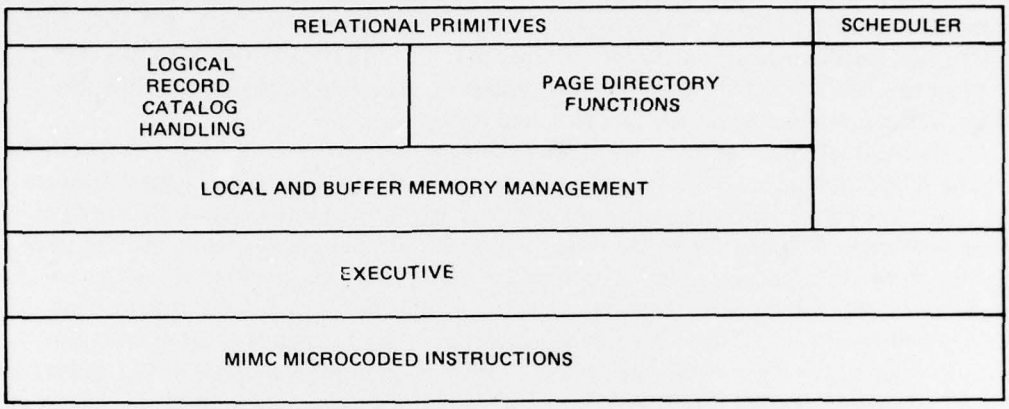


Figure 3. Data processor hierarchy of functions.

THE SCHEDULER

The scheduler is designed to be in an idling mode which continually examines a queue of processes ready to be executed. If the ready queue is empty, idling continues; otherwise the first process in the queue is initiated. An executing process (which may consist of several activities for the executive) may run to completion or be suspended prior to completion either because the process requires a resource which is not currently available or because the process is interrupted. Interruptions can occur by a buffer-memory controller request, by completion of a data transfer, or by an alert. The scheduler processes the interrupt and, if appropriate, determines if some other process is now ready (eg, a process that was suspended waiting for data from disk) for execution. If so, then the priority of the now ready process is compared with that of the interrupted process, the process with the higher priority is executed, and the one with the lower priority is inserted in the appropriate ready-queue position. When a process is completed, the highest priority process awaiting completion is placed in the ready queue.

DATA PROCESSOR EXECUTIVE CYCLE

The philosophy of executive cycle operation is based upon an activity discipline in which activities are executed in sequence from an activity tree. An activity-tree organization is used rather than an activity list, since it is desirable to move grouped activities on and off the tree, an action which is quite difficult to perform with a list structure. The sensing of input from the user processor is always on the activity tree with appropriate inputs which

result in placing other activities on the tree. However, an input does not place an activity directly on the tree, since such an action could result in the loss of the executive system's integrity. Rather, the executive looks for a flag set by inputs on the input list on the activity tree. If a flag is found, a different activity is executed. The executive is said to scan the tree if it executes each activity on the tree once. The tree actually consists of both a slow and a fast tree, but for the present purposes, the trees together will be referred to as "the tree." The executive first scans the entire fast tree, unless empty, then executes one slow tree activity, returns to the scan of the fast tree, and so forth.

Each activity can call four executive functions: Exit, which allows the executive to continue scanning the tree but leaves that activity on the tree; Quit, which also continues executive scanning of the tree but also removes the activity from the tree so that activity is not encountered on the next scan; Activate, which initiates new activities by placing new branches on the tree (requires the location of the new branch be specified as well as which tree (fast or slow) will be used); and Retire, which is the complement of Activate. This function removes an entire branch from the activity tree and, when it is carried out, the executive does not continue with the ongoing scan but reinitiates the scan of the entire tree.

The order in which activities are executed cannot be assumed, but, if an order is required, a sequence of "activate" and "quit" functions within the activities is necessary.

This architecture was selected for the executive since it supports rule-based programming as well as alerts, both of which are important in the intended application. The architecture does not exclude sequential programming since an activity is a sequential program. In addition, the architecture contains the spirit of timesharing since a wait in one sequential program can be utilized by another program. Slicing can also be introduced if required.

RELATIONAL DATA-MANAGEMENT OPERATORS

The logical-record-catalog-handling utilities are used to support the implementation of the relational data-management operators. These relational operators include mapping, prime-symbol mapping, mapping-composition, projection, create-relation, delete-relation, insert-new-row, delete-row, and replace-row (see the description of the SQUARE Data Sublanguage in reference 5).

The mapping operator returns a new relation of values in the range column or columns of the given relation whose associated domain column(s) values match the domain argument of the operator. The domain argument can itself be a relation.

Prime-symbol mapping is the same as mapping except that all duplicate elements are maintained in the new relation. This is useful when element counts, sums, and averages are desired.

Mapping-composition is an operation which takes the result of the first mapping it receives and uses that relation as the argument for the second mapping. Of course, the domain of the relation of the second mapping must be the same type of relation as the value of the first mapping. Mapping-composition, thus, calls on the first mapping. The first mapping will form as a value the logical record address of a new temporary relation. When the first mapping is complete, mapping-composition then calls the second mapping with the argument being the value of the first mapping, specifically its logical record address.

---

5. Boyce, R, Chamberlin, D, King, W, and Hammer, M, "Specifying Queries as Rational Expressions: The SQUARE Data Sublanguage," Communications of the Association for Computing Machinery, November 1975

At this point, no real constraints have been placed on the domain argument of a mapping. As previously mentioned it could be a relation, but it can also be a relation with constraints; ie, each ship of a specific task group which has 30 percent of its fuel remaining. Here, percentage of fuel remaining on a ship is a relation, but the mapping will only pick out those ships having 30 percent of their fuel remaining. The mapping operator and its variations are designed to be general enough to handle calculable arguments as well as lists of arguments (or an argument which is a relation).

Projection returns as its value a given column or columns of a relation, and is most useful in composition with mappings. Projection is implemented as a trivial case of mapping where the domain argument is all elements.

Create-relation determines the logical record address of the relation and then adds an element on the first page of that record which indicates the type of relation. For example, the number of columns will be given as will the meaning of each of the columns. Delete-relation returns the pages of its logical record to the available page list.

Insert-new-row and delete-row perform the normal operations. Replace-row replaces (or updates) the row of the relation. The requested update can be an arithmetic manipulation on each of the elements of the record or selected elements of the record. It can also be a strict replacement of selected elements.

BULK MEMORY PAGE MANAGEMENT

The bulk memory (currently disk) is organized into pages of fixed length with pages grouped into logical records. A pointer to the head of each logical record is maintained as an index to bulk storage by use of the logical record catalog.

The next layer of indices into bulk storage is maintained in the page directory, where the connectivity of each page to other than itself is maintained. Pages in use are kept in this directory and pointers to the rest of the pages are located in the availability list.

A unique feature of the system is the change record in which changes to pages are recorded until such time as the changes are actually performed in bulk storage. This feature eliminates excessive waiting for access to bulk storage when a user requests an update. Change-record functions are implemented to effect one change at a time on a noninterference-with-the-user basis.

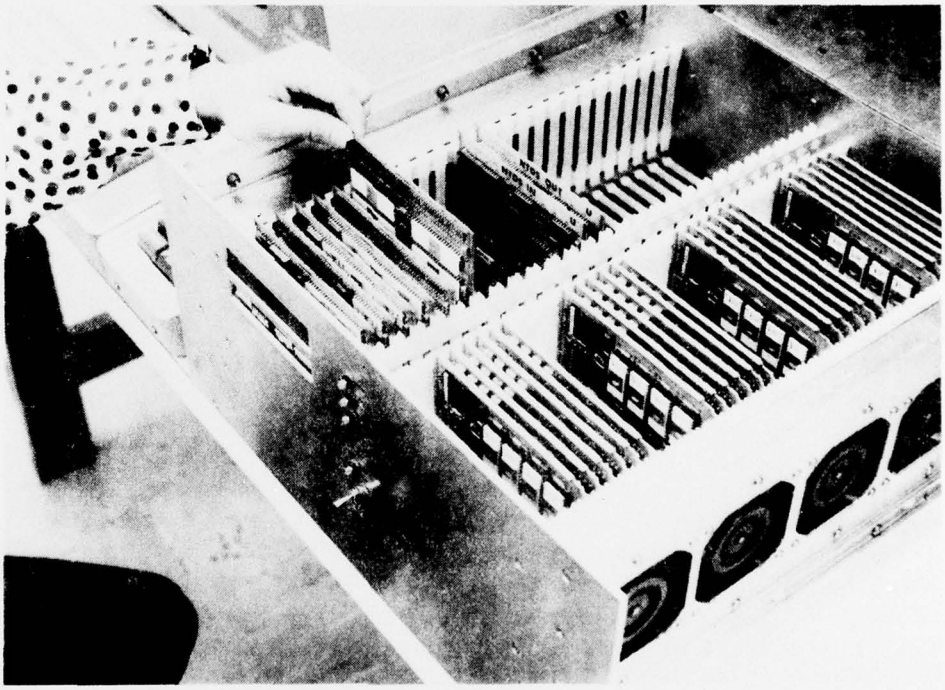
The handlers and utilities are designed to provide redundancy in the pointer structure and to provide additional means for finding those pages belonging to a relation or logical record without loading all pages of the record. It should be noted, however, that the next-page-pointer redundancy for each page of the record is maintained in order to avoid loss of record structure if the page directory and/or catalog becomes lost.

BUFFER AND LOCAL-MEMORY MANAGEMENT

These functions are designed for maintaining memory status with respect to space available for pages. Bits are kept for each page location in buffer and in local memory to show if the page is locked into memory and cannot be written to disk, if the page is protected against writing, has been written on, and so forth. One of the functions copies pages from disk to buffer and/or local memory as space is available.

MEDIA INDEPENDENT MEMORY CONTROLLER (MIMC)

The data processor hardware and firmware design is based on that of the MIMC processor, a processor being designed to interface the newer solid-state technologies, such as charge-coupled devices, with advanced systems which will require storage of digital data (fig 4). The central processor used is a general purpose, four-bit-slice microprocessor which is the equivalent of the Monolithic Memories 6701. This is a bipolar chip which can be cycled in about 300 nanoseconds, allowing high transfer rates and minimum processing time for the required system functions such as searching for information, updating, and data analysis.



LSF 0006-01-77

Figure 4. Media Independent Memory Controller (MIMC).

The present word width is 32 bits for high speed and compatibility with the AN/UYK-7 and other existing Navy computers which will still be in use for the next 5 to 10 years. Maximum memory size is 65 thousand words.

BASIC INSTRUCTION FORMAT. The lower half word is a 16-bit address. The upper half contains an 8-bit operation code and two 4-bit register addresses. The repertoire has not been completely defined at this time but the microprogrammability of the controller allows the instructions to be modified rather simply. A basic set including load and store, exchange, shifts, and basic logical relational, arithmetic, jumps, and stacking instructions has

been defined. Other instructions have been added to localize all data access, to decrement a cell conditionally, to initialize a repeat flag, to save registers on a subroutine call and restore them on a return, to do a binary search, and to do a bubble sort. Hardware has been designed to compress and expand fields of data.

The internal architecture of MIMC uses a bidirectional bus with bus control signals similar to the controls of the SEM/SSIXS 8080 (NELC Technical Note 3005, A Quick and Easy Design (QED) Terminal – INTEL 8080 Microprocessor and Common-Bus System, by GR Huckell, 18 July 1975).

The interrupt structure allows 32 interrupt sources which can be enabled or disabled under software control. Priority is determined by a daisy-chained technique; the highest priority disables the lower priorities. When an interrupt is received and accepted by the processor, the starting address for the particular interrupt handling routine is forced into the bus and the processor executes the required program.

The input/output (I/O) philosophy uses the normal addressing procedure to designate ports: regards an I/O port as a read or write memory address. For systems having only a very few ports, a separate dedicated control line can be used to simplify address decoding on the I/O port module. Input-output interfaces currently available include the 32-bit Navy Tactical Data System parallel interface and the RS-232 Teletype interface.

INTERFACE PROCESSOR

This processor is designed to accommodate limited-syntax (such as fixed-field records) bulk updates which can arrive from a variety of different computer I/O channels and/or data links. Thus, the processor can readily accommodate different word lengths and data rates. All updates will be done eventually in bulk storage through the cache-memory system but under the control of the data processor.

AUXILIARY MEMORY SYSTEM

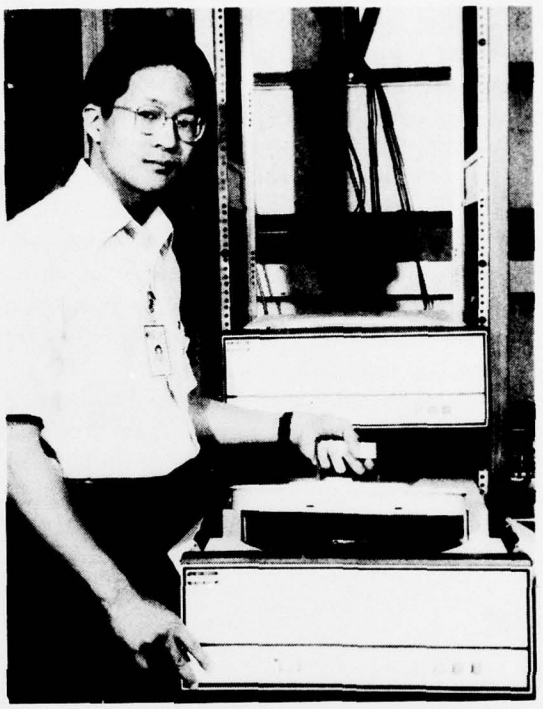
Most requirements for Navy information storage and retrieval demand random-access auxiliary storage. An initial choice for such a storage device is the Model 44 Diablo disk drive with a nominal disk capacity of 6 250000 sixteen-bit words and an access time of about 50 milliseconds (fig 5). This drive is reasonably typical of the lower-cost disk drives which are available.

The functions of the disk controller and the I/O channel interface will be performed by a combination controller and buffer-memory (12000 thirty-two-bit words) designed and constructed at NELC (fig 6).

There are a number of advantages in having a buffer-memory interface between a computer and a disk drive. These include a reduction in the total disk-drive access time and the provision of a temporary storage area and interface for multiple devices.

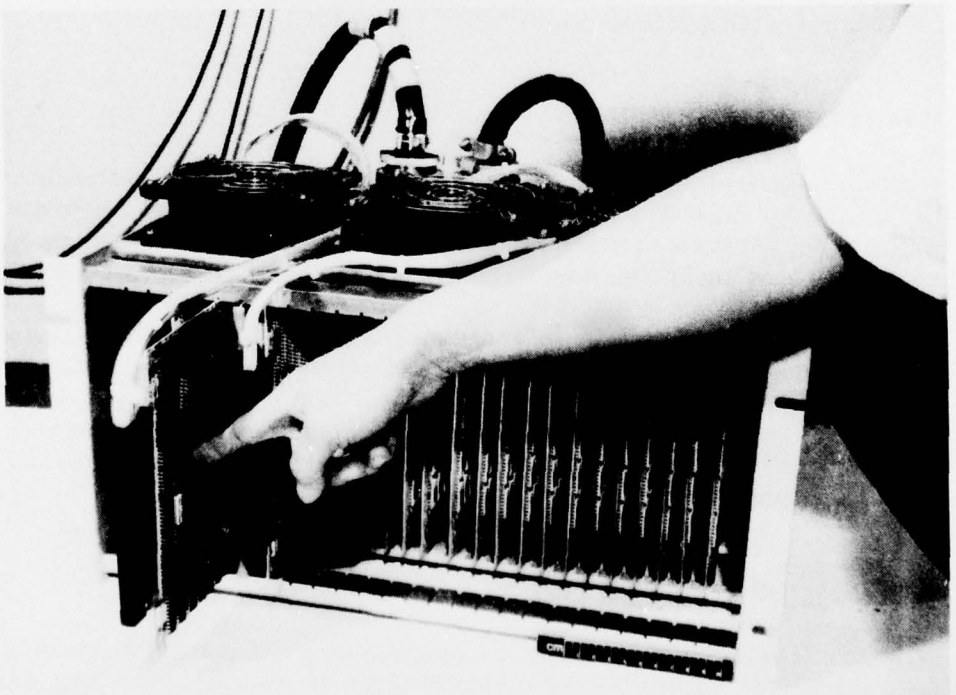
REDUCTION OF ACCESS TIME

When using a disk drive, head positioning and latency of the disk drive (waiting for the disk drive to reach the proper track and the sector within the track) present a delay problem. Without the buffer-memory (a direct interface between the computer and the disk



LSF 0856-05-76

Figure 5. Diablo disk-drive auxiliary memory system.



LSF 0859-05-76

Figure 6. Combination controller and buffer memory.

drive), the computer would have to wait the length of this disk-access time which is about 50 milliseconds for the Diablo disk drive. With the buffer-memory interface, the computer instructs the main controller to input a certain page into the buffer. While the main controller waits to access the disk and inputs the page to the buffer, the computer can be processing other data and can receive an interrupt whenever the data are in the buffer. The 50-millisecond access time of the Diablo disk drive is very significant and will slow the processing time of the computer if the disk is interfaced directly.

TEMPORARY STORAGE AREA

The buffer can be used to store intermediate data of an incomplete operation (temporary scratchpad memory). The buffer will allow small changes to be made in a page from the disk without having to bring a whole page of memory into main memory. The page is first brought from disk to the buffer and then the changes are sent from the computer to the desired location in the buffer. As few computer words as one may be rewritten from the buffer back to the disk. This saves computer I/O time and main-memory space. Page thrashing can be reduced with the use of the buffer as a temporary storage when generating and recognizing item names (especially on a small-memory computer).

INTERFACE FOR MULTIPLE DEVICES

The buffer will be used to interface more than one device to each other and to the disk drive (fig 1). The buffer presently has four ports. In the proposed ISAR architecture, two ports are devoted to the data processor and interface processor microcomputers and the third is devoted to the Diablo disk drive. The fourth port is not used at present.

REFERENCES

1. Canady, RH, et al, "A Back End Computer for Data Base Management," Communications of the Association for Computing Machinery (CACM), October 1974

2. California Institute of Technology REL Report 3, REL – An Information System for a Dynamic Environment, by BH Dostert, December 1971

3. California Institute of Technology REL Report 4, Computer System Support for Data Analysis, by NR Greenfield, March 1972

4. California Institute of Technology REL Report 17, The REL Paging System, by FB Thompson, 1974

5. Boyce, R, Chamberlin, D, King, W, and Hammer, M, "Specifying Queries as Rational Expressions: The SQUARE Data Sublanguage," Communications of the Association for Computing Machinery, November 1975

6. Naval Electronics Laboratory Center Technical Note 3005, A Quick and Easy Design (QED) Terminal – INTEL 8080 Microprocessor and Common-Bus System, by GR Huckell, 18 July 1975*

*NELC Technical Notes are informal documents intended primarily for use within the Center.

INITIAL DISTRIBUTION LIST

NAVAL ELECTRONIC SYSTEMS COMMAND
- NELEX-330 (R. KAHANE)
- NELEX-330 (C. STOUT) (5)
- NELEX-330 (J. MACADO)
- NELEX-330 (R. FRATILLA)
- NELEX-570 (R. DUBELOIS)
- PME-108 (D. SCHUTZER)
- PME-108 (J. OLSON)
- PME-108 (J. NEWELL)
- PME-108 (T. CONNELLY)
- PME-108 (G. NEELEY)
- PME-108 (D. MULLIKIN)
- PME-108 (G. HAMILTON)
- PME-108 (S. DAVIDSON)

DEFENSE TELECOMMUNICATIONS AND COMMAND AND CONTROL SYSTEMS
- F. KUO

DEFENSE COMMUNICATIONS ENGINEERING CENTER COMMAND AND CONTROL TECHNICAL CENTER
- M. CHAMPAIGN (2)
- LTC T. H. BAUMGARTNER (2)
- BOB MARION (2)

OFFICE OF NAVAL RESEARCH
- ONR-437 (J. TRIMBLE)
- ONR-437 (M. DENICOFF)
- ONR-437 (G. GOLDSTEIN)

ROME AIR DEVELOPMENT CENTER
- DUANE STONE (3)
- PAT LANGENDORF

NAVY PERSONNEL RESEARCH AND DEVELOPMENT CENTER
- J. WOLFF (2)

NAVAL UNDERSEA CENTER
- CODE 14 (C. MERROW) (2)

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
- IPTO (CDR F. HOLLISTER)

SYSTEM DEVELOPMENT CORPORATION
- SANTA MONICA, CA 90406
  - GEORGE CADY
  - JEFF BARNETT

CALIFORNIA INSTITUTE OF TECHNOLOGY
- PASADENA, CA 91103
  - DR. F. THOMPSON

IBM RESEARCH LIBRARY
- SAN JOSE, CA
  - VI MA

DEFENSE DOCUMENTATION CENTER (12)