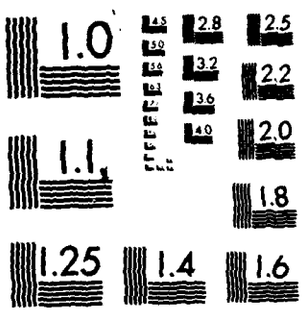
AD-A139 385 ADVANCED DOCUMENT RETRIEVAL SYSTEM(U) CONTROL DATA CORP 1/3
MINNEAPOLIS MN W CYRE ET AL. JUL 83 RADC-TR-83-168
F30602-79-C-0231

UNCLASSIFIED F/G 9/2 NL

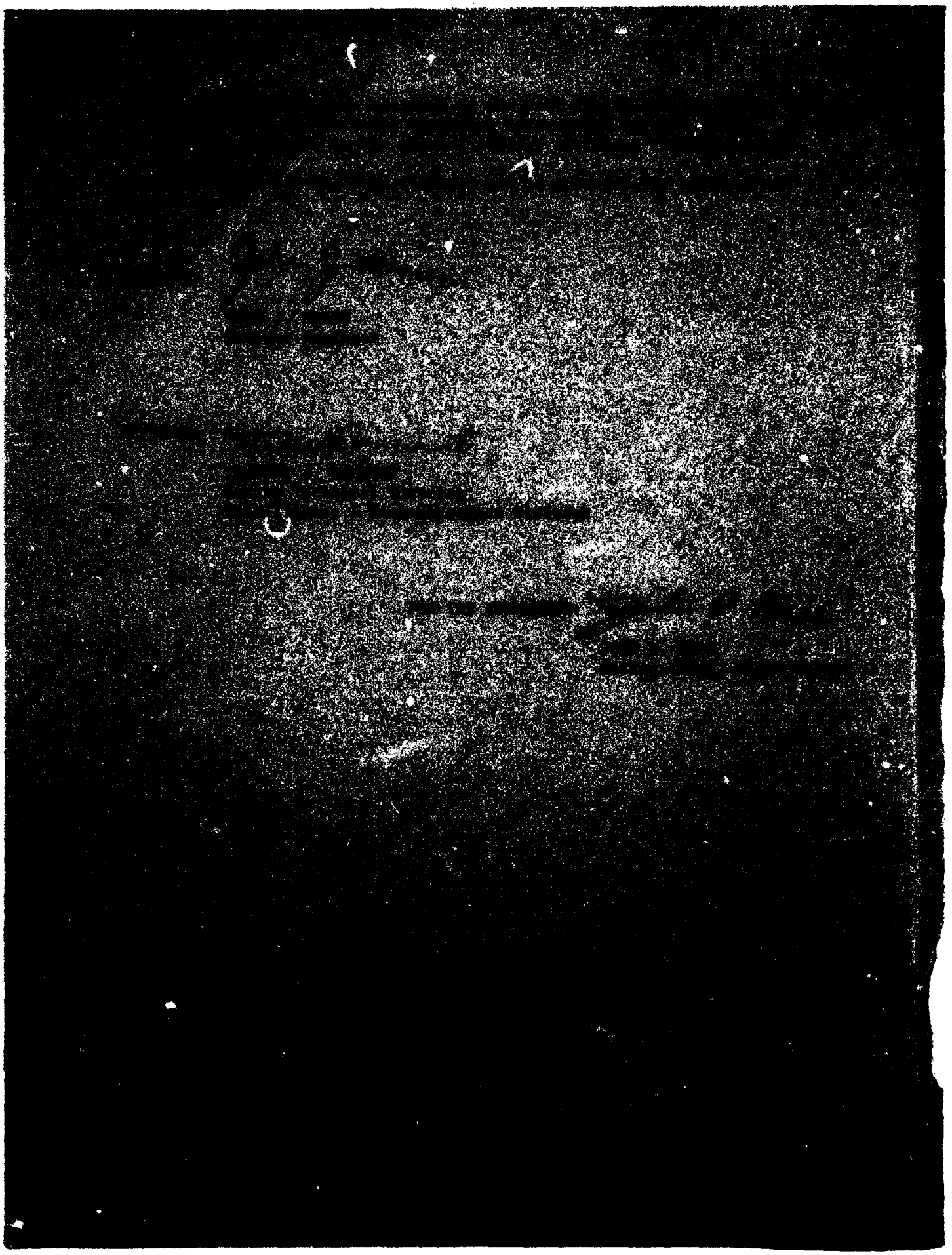


[redacted]

[redacted]

[redacted]

[redacted]



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-83-168	2. GOVT ACCESSION NO.: AD-A135 385
	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): ADVANCED DOCUMENT RETRIEVAL SYSTEM	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report Sep 79 - Apr 83
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Walling Cyre, Joseph Vaughan, Steven Adkins	8. CONTRACT OR GRANT NUMBER(s): F30602-79-C-0231
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Control Data Corporation, 2800 East Old Shakopee Road, Minneapolis MN 55420	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 62441630
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (IRDT), Griffiss AFB NY 13441	12. REPORT DATE: July 1983
	13. NUMBER OF PAGES: 280
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: John J. Maier (IRDT)

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

Document Retrieval
Associative Processing
Processor Array

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

This work is related to a previous study, performed under RADC contract F30602-78-C-0065, in which the intelligence retrieval application and data bases at FTD were analyzed; potential usefulness of reconfigurable arrays and associative memories was assessed; and an exploratory model of an associative unit was designed, fabricated, and evaluated. (Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The work performed under this effort extended the analysis, acquired additional data and developed and demonstrated a full scale advanced document retrieval system. Models were developed for text compression, data base partitioning trade-offs were made, and analysis was performed for automatic data base generation.

The work included the design and fabrication of a full scale associative processing unit (AU) with 1000 MOPS processing capability and 256K-bit storage capacity. This was interfaced to an array of three existing Advanced Flexible Processors (AFP), a host computer, and a secondary mass storage system. This equipment configuration served as the hardware basis for the document retrieval demonstration.

The successful demonstration of the Advanced Document Retrieval System included:

- o Data base update
  - hexical decomposition
  - sorting
  - indexing
  - file formation
- o Multiple concurrent data bases and merging
- o Full set of search operators
- o Interpretation of complex search logic statements
- o Interactive user query and display functions

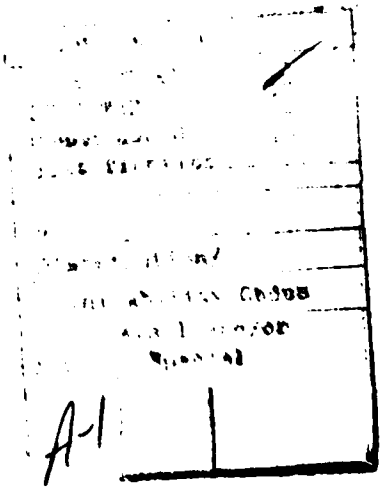
An analysis of AU and AFP performance based on a system model supporting 100 simultaneous users each entering one command per 10 seconds with a data base of 350 million document tokens, and a monthly update of 1,750,000 tokens was performed, using timing derived from the demonstration software. A single AFP, processing various search and update functions yielded performance exclusive of disc access time, ranging from 40 to 7000 times faster than real-time requirements. Similarly the AU performed its assigned functions 13,000 times faster than real-time.

FINAL REPORT SUMMARY

This report describes a study and demonstration performed by the Information Sciences Division of Control Data Corporation for Rome Air Development Center under contract F30602-79-C-0231. The objective of this effort was to demonstrate highly useful search strategies for retrieval of intelligence information, exploiting the computational power of an array of microprogrammable processors and the search speed of an associative memory.

This work is related to a previous study, performed under RADC contract #F30602-78-C-0065, in which the intelligence retrieval application and data bases at FTD were analyzed; potential usefulness of reconfigurable arrays and associative memories was assessed; and an exploratory model of an associative unit designed, fabricated, and evaluated.

The work performed under this effort extended the analysis, acquired additional data, and developed and demonstrated a full-scale advanced document retrieval system. The effort consisted of four general categories of activities. The first activity was associated with development of the document retrieval model. This involved extending the analysis of the operational intelligence activities characterizing the data base and its usage, and extending the demonstration data base to 3000 documents. Models were developed for text compression, data base partitioning trade-offs were made, and analysis was performed for automatic data base generation.



The second category of work included the design and fabrication of a full-scale associative processing unit with 1000 MOPS processing capability and 256K-bit storage capacity. This was interfaced to an array of three existing Advanced Flexible Processors, a host computer, and a secondary mass storage system. This equipment configuration provided a highly flexible, very high speed processing complex which served as the hardware basis for the document retrieval demonstration.

The third category of effort was the development of the software to implement, operate, and support the advanced document retrieval system. This software was designed to implement the algorithms and processes defined as a result of the analysis and was partitioned among the hardware units to most effectively exploit their characteristics. The software developed in this effort included:

- o Microcode for the Associative Processor Unit.
- o Microcode for the AFP's.
- o Software for the host, DEC 11/70, which provides overall system control and processes operator queries.
- o Controlware for the secondary storage controller.

The fourth area of effort included the successful demonstration of the Advanced Document Retrieval System including:

- o Data base update
  - lexical decomposition
  - sorting
  - indexing
  - file formation
- o Multiple concurrent data bases and merging
- o Full set of search operators
- o Interpretation of complex search logic statements
- o Interactive user query and display functions

Analysis of AU and AFP performance based on a system model supporting 100 simultaneous users each entering one command per 10 seconds with a data base of 350 million document tokens and a monthly update of 1,750,000 tokens was performed, using timing derived from the demonstration software. A single AFP, processing various search and update functions yielded performance, exclusive of disc access time, ranging from 40 to 7000 times faster than real-time requirements. Similarly the AU performed its assigned functions 13000 times faster than real-time. The demonstration system performance was limited principally by the speed of available disc storage units.

The evaluation data obtained on the project was used to define a production system configuration based on a single AFP capable of supporting up to 1000 users interactively.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

LIST OF FIGURES

L I S T O F T A B L E S

1.0 INTRODUCTION

This report describes a study and demonstration performed by the Information Sciences Division of Control Data Corporation for Rome Air Development Center under contract F30602-79-C-0231. The objective of this effort was to demonstrate highly useful search strategies for retrieval of intelligence information, exploiting the computational power of an array of microprogrammable processors and the search speed of an associative memory.

This work is related to a previous study, performed under RADC contract #F30602-78-C-0065 and reported in RADC-TR-79-313, in which the intelligence retrieval application and data bases at FTD were analyzed; potential usefulness of reconfigurable arrays and associative memories was assessed; and an exploratory model of an associative unit designed, fabricated, and evaluated.

The work performed under this effort extended the analysis, acquired additional data, and developed and demonstrated a full-scale advanced document retrieval system. The effort consisted of four general categories of activities. The first activity was associated with development of the document retrieval model. This involved extending the analysis of the operational intelligence activities characterizing the data base and its usage, and extending the demonstration data base to 3000 documents. Models were developed for text compression, data base partitioning trade-offs were made, and analysis was performed for automatic data base generation.

The second category of work included the design and fabrication of a full-scale associative processing unit with 1000 MOPS processing capability and 256K-bit storage capacity. This was interfaced to an array of three existing Advanced Flexible Processors, a host computer, and a secondary mass storage system. This equipment configuration provided a highly flexible, very high speed processing complex which served as the hardware basis for the document retrieval demonstration.

The third category of effort was the development of the software to implement, operate, and support the advanced document retrieval system. This software was designed to implement the algorithms and processes defined as a result of the analysis and was partitioned among the hardware units to most effectively exploit their characteristics. The software developed in this effort included:

- o Microcode for the Associative Processor Unit.
- o Microcode for the AFP's.
- o Software for the host, DEC 11/70, which provides overall system control and processes operator queries.
- o Controlware for the secondary storage controller.

The fourth area of effort included the successful demonstration of the Advanced Document Retrieval System including:

- o Data base update
  - lexical decomposition
  - sorting
  - indexing
  - file formation
- o Multiple concurrent data bases and merging
- o Full set of search operators
- o Interpretation of complex search logic statements
- o Interactive user query and display functions

Analysis of AU and AFP performance based on a systme model supporting 100 simultaneous users each entering one command per 10 seconds with a data base of 350 million document tokens and a monthly update of 1,750,000 tokens was performed, using timing derived from the demonstration software. A single AFP, processing various search and update functions yielded performance, exclusive of disc access time, ranging from 40 to 7000 times faster than real-time requirements. Similarly the AU performed its assigned functions 13000 times faster than real-time. The demonstration system performance was limited principally by the speed of available disc storage units.

The evaluation data obtained on the project was used to define a production system configuration based on a single AFP capable of supporting up to 1000 users interactively.

This report is organized into six sections:

Section 1 - Summarizes the project activity and results.

Section 2 - Analyzes the document retrieval and message handling problem as applicable to FTD, develops algorithms and sizing estimates for the processes to be supported and develops a processing model for the Advanced Document Retrieval System demonstration.

Section 3 - Describes the equipment used in the demonstration.

Section 4 - Describes the software developed to implement the processing model described in Section 2 for the demonstration.

Section 5 - Describes the method by which the Advanced Document Retrieval System was demonstrated.

Section 6 - Provides an analysis of the system performance, suggests a production document retrieval system configuration, and makes recommendations for further study.

Additional details on the Advanced Document Retrieval System are provided in the five appendices:

A - Demonstration Document Retrieval System Users Manual

B - AFP Description

C - Associative Unit

D - System Software Description

E - Annotated A.U. Microcode

2.0 ANALYSIS OF THE APPLICATION

2.1 INTRODUCTION

This section deals with the characterization, decomposition, and computational support requirements of document retrieval and message handling systems used in military intelligence activities. Document retrieval systems are employed to maintain a data base of document texts which may be selectively and interactively retrieved by intelligence analysts through the specification of words or terms of interest. It is also required that these systems automatically dispatch copies of newly added documents to analysts based on their previously specified mission profiles. The primary function of message handling systems is to automatically route intelligence messages to intelligence analysts based on their mission profiles. It is also necessary to maintain a repository of recently received messages for interactive retrieval. Thus, the functions of document retrieval and message handling systems are the same. The primary distinction between the two lies in the priorities of the retrieval and routing functions and the magnitude of the data base.

A very general model for document retrieval and message handling systems is presented in Figure 2-1. The documents or messages enter a system through an Update process. This process performs two functions. First, a collection of documents is processed to prepare a temporary data base which contains the collection and conforms to the structure of the main data base of documents.

The Update process then applies the mission profiles for the analysts to the temporary data base in order to prepare a mailing list used by the Dispatch process. Finally, the temporary data base is integrated with the main data base by cataloging it or merging it with an existing data base component. (The structure of the main data base is discussed in Section 2.4.)

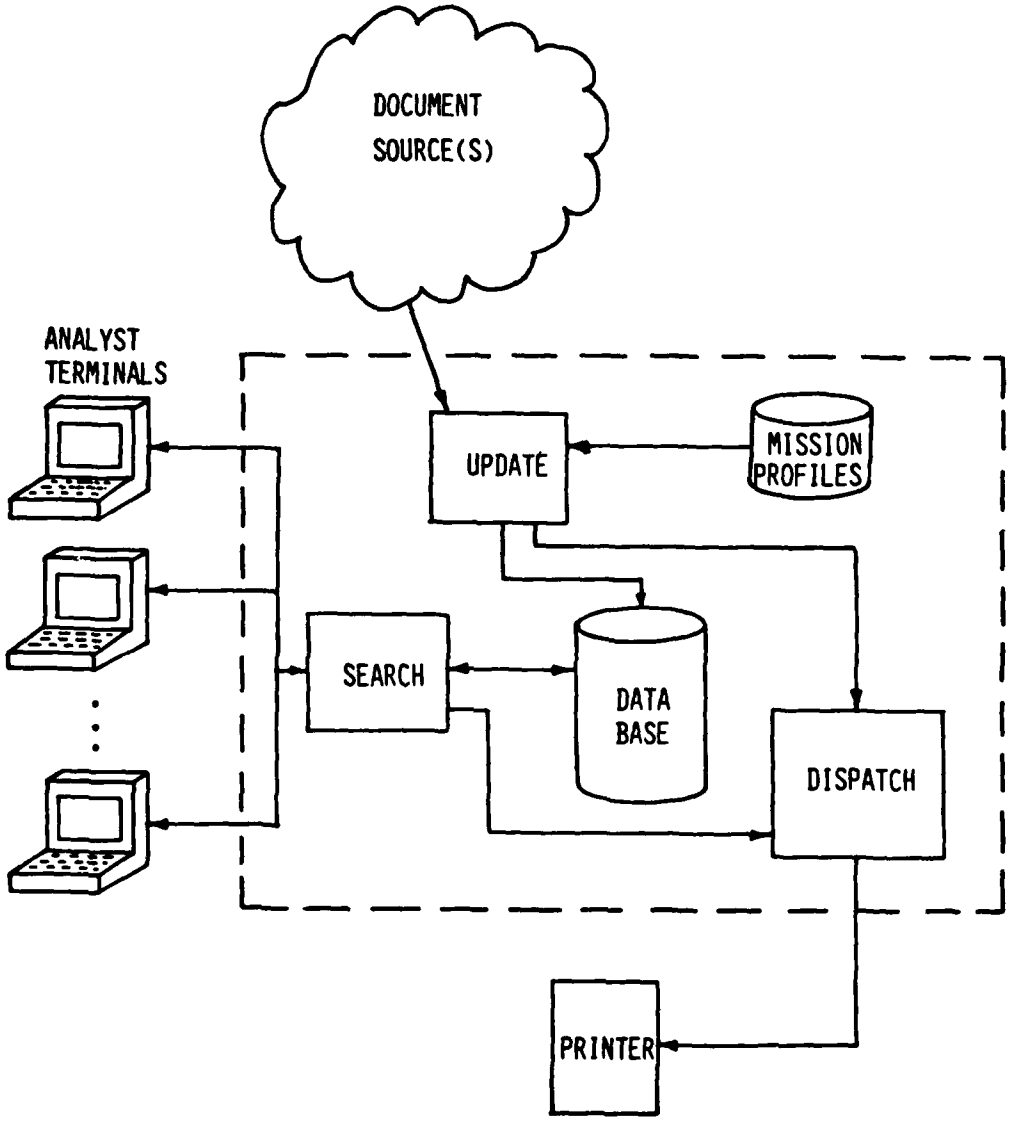


Figure 2-1. A General Model for Document Retrieval and Message Handling Systems

The analysts interactively search the main data base and select documents to be dispatched to them using the Search process. The Search process allows analysts to form subsets of documents using expressions whose operands are words or terms occurring in the documents or are references to sets of documents. The Search process also allows the analysts to interactively examine or browse through a set of documents.

The Dispatch process merely distributes copies of documents based on the analyst names or mail codes associated with the document sets formed by the Search and Update processes. Because dispatching is simple and rather peripheral to the main tasks, little further mention of it will be given in this report.

Before continuing the decomposition of the document retrieval and message handling processes, attention is given in the next sections to the characteristics of the source documents and the requirements of the intelligence analysts.

2.2 CHARACTERISTICS OF THE DOCUMENTS

While the documents found in document retrieval systems differ significantly in content and function from those of message handling systems, their structures or forms are essentially the same with respect to mechanical processing. For the purposes of this report, a document consists of several, variable length zones. Many types of zone may occur, each with its own semantics and syntax. Examples of document zones are those used for an identification number, the authors' names, the source, a date, a classification level, and paragraphs of text. For current purposes, however, all zones are viewed as paragraphs of text.

A paragraph (zone) consists of one or more sentences, which, in turn, are strings of words or lexical units. In simplified form, paragraphs are identified by special markers including a paragraph (zone type) number. Sentences are terminated by periods, and tokens (words) are delimited by any special characters excluding the apostrophe, hyphen, and slash. This simple document model is given formally in Figure 2-2.

Beyond the intelligence activities analyzed during this project, the principal differences in document characteristics are the lengths of the documents and the vocabularies of the documents. Of these, only the document lengths are of interest here. Differences were also seen in the quantities of documents in the data base and the arrival rates of new documents. A summary of the observations obtained by computer analysis of a document set, and discussions with intelligence administrators, support specialists, and contractors appears in Table 2-1. (Some of these figures represent quantities desired by the intelligence activities.)

TABLE 2-1. DOCUMENT CHARACTERISTICS

Characteristics	Document Retrieval	Message Handling
Average Document Length (words)	70	1,000
Number of Documents Represented in Data Base	5,000,000	50,000
Arrival Rate	25,000 weekly	1 per min. (peak)

<document> ::= <paragraph> [<paragraph>]

<paragraph> ::= <zone type> <sentence> [<sentence>]

<sentence> ::= <token> [<token>].

<token> ::= <character> [<character>] <non-character>

<character> ::= A | B | ••• | Z | 0 | 1 | ••• | 9 | - | ' | /

NOTE

[ ] denotes one or more repetitions of the enclosure

| denotes alternatives

Figure 2-2. Syntax of a Simple Document Model

2.3 Characteristics of the Users

The users of the document retrieval and message handling systems reviewed during this project are intelligence analysts. Each analyst has a mission to monitor one or more activities. Depending on the agencies involved, missions may vary from monitoring medical research in a foreign principality to tracking the movement of ships. The systems studied support these missions by evaluating incoming documents or messages with respect to profiles of the missions and by providing for interactive searching of documents in the data base.

The user interacts with the systems through a command language. The specific command language supported by the system developed during this project is described fully in a later section, but for current purposes, only the search type of command is necessary.

Search commands are expressions used to specify subsets of the documents contained in the data base. The simplest form of a search expression is a token or word which specifies the set of all documents containing at least one instance or occurrence of that token. Another search expression or form is the truncated token, which is denoted by a token ending in a dollar sign ($) possibly followed by a digit. For example, the expression 'AIR$3' specifies the set of documents having one or more occurrences of any token of three to six characters, the first three of which are 'AIR'. Leaving off the digit following a $ accepts tokens of any number of characters following the specified prefix. Another basic expression is a reference to a previously entered expression, and is denoted by the search command numbers assigned by the system. An expression reference indicates the set of documents specified by the given expression.

More complex search expressions can be formed using a set of logical operators. The simplest operator is 'NOT' which specifies the complement of the set of documents specified by the expression following the operator. The 'OR' and 'AND' operators specify the union and intersection of document sets, respectively. The expressions 'LASER ADJ WEAPONS' and 'LASER WEAPONS' specify documents containing any occurrence of the token 'WEAPONS' which is immediately preceded by an occurrence of 'LASER'. The forms 'A SEN B' and 'A PAR B' denote documents containing the specified tokens in the same sentence and paragraph, respectively. A simplified syntax for search expressions is given in Figure 2-3. The values returned immediately to the user as the result of executing a search command are counts of the number of documents in the result set and the number of instances (occurrences) throughout those documents which satisfy the expression. The user can then construct the next search command to either restrict or augment the previous set of documents.

The mission profiles of the analysts mentioned earlier are merely stored sequences of search commands, and are automatically applied to incoming documents.

Quantitative information about users was obtained through the analysis of a recording of terminal transaction and a set of analyst profiles provided by FTD on their unclassified document retrieval system. Most of the data was obtained from the profile tape, which contained almost 100,000 statements. These statements were analyzed to determine the types of search operands (simple expressions) typically used, the distribution of operator types, and dominant forms of search expression used. Results of this analysis are given in Tables 2-2 through 2-4. In the analysis of search forms, it was observed that subexpressions consisting of words connected by OR operators and of references connected by OR operators occurred very frequently.

```
<expression>  ::= <token>

                  <token>  $

                  <token>  $  <digit>

                  <reference>

                  not <expression>

                  (<expression>)

                  <expression> <operator> <expression>

                  <expression> <expression>

<operator>    ::= OR | AND | ADJ | SEN | PAR

<reference>   ::= <digit>  [  <digit>  ]
```

Figure 2-3. A Syntax for Search Expressions

The primary result obtained from the analysis of transactions between users and the system was the distribution of user response times shown in Figure 2-4. The user response time is the time it takes a user to formulate and enter a command measured from the system's response to the user's previous command. As can be seen, most of the time users responded in 5 to 15 seconds.

Based on the information in this section, it is possible to derive some very useful system requirements. From Figure 2-4 it is reasonable to expect the system to process one search command from each active user per ten seconds. Then, with 100 users searching in a time shared manner, the system must satisfy one command per 100 milliseconds. This would also give the system a very satisfactory response time.

From the analysis of 100,000 search commands on a profile data set, the average number of search operands per command is 3.4, and the average operators per command is 2.5 (Tables 2-2 and 2-3). Of the search operands 70% are tokens, giving 2.4 tokens per command, or 0.24 tokens per user command per second. These requirements are summarized in Table 2-5.

2.4 Structure and Characteristics of the Data Base

The major components of a document data base are those containing the documents and the index to the documents. An example of the contents of these two components is shown in Figure 2-5. The document component is implemented as a single sequential file. Because the index component is used in real-time searching, its implementation is more complex in order to improve performance of a sequential file.

TABLE 2-2. TYPES OF SEARCH OPERANDS

Operand Type	Percentage of Use*	
Tokens		70%
Words	54%	
Truncations	16%	
Statement References		30%

*Sample of 340,000 operands

TABLE 2-3. TYPES OF SEARCH OPERATORS

Operator Type	Percentage of Use*
NOT	0.2
OR	80
AND	2
ADJ	3
SEN	13
PAR	2

*Sample of 250,000 operators

TABLE 2-4. FREQUENT SEARCH EXPRESSION FORMS

FORM *	Frequency (percent)
w SEN w OR w	7.1
r AND r	5.0
w SEN w	4.8
w	4.7
r OR r OR r OR r OR r OR r OR r	3.8
r PAR r	3.0
r OR r	2.8
r OR (w OR w OR w OR w)	2.4
w ADJ w	1.8
w SEN (w OR w OR w)	1.8
t	1.7
w OR w OR w OR w OR w	1.7
r OR r OR r	1.4
w SEN t	1.3
r SEN r	1.1
r ADJ w ADJ r	1.0
TOTAL	45.4

* w denotes word
  t denotes truncated word
  r denotes statement reference

TABLE 2-5. SYSTEM REQUIREMENTS PER ACTIVE USER

Search Commands per second	0.10
Search Tokens (words or truncations) per second	0.24
Statement References per second	0.10
Search Operators per second	0.25

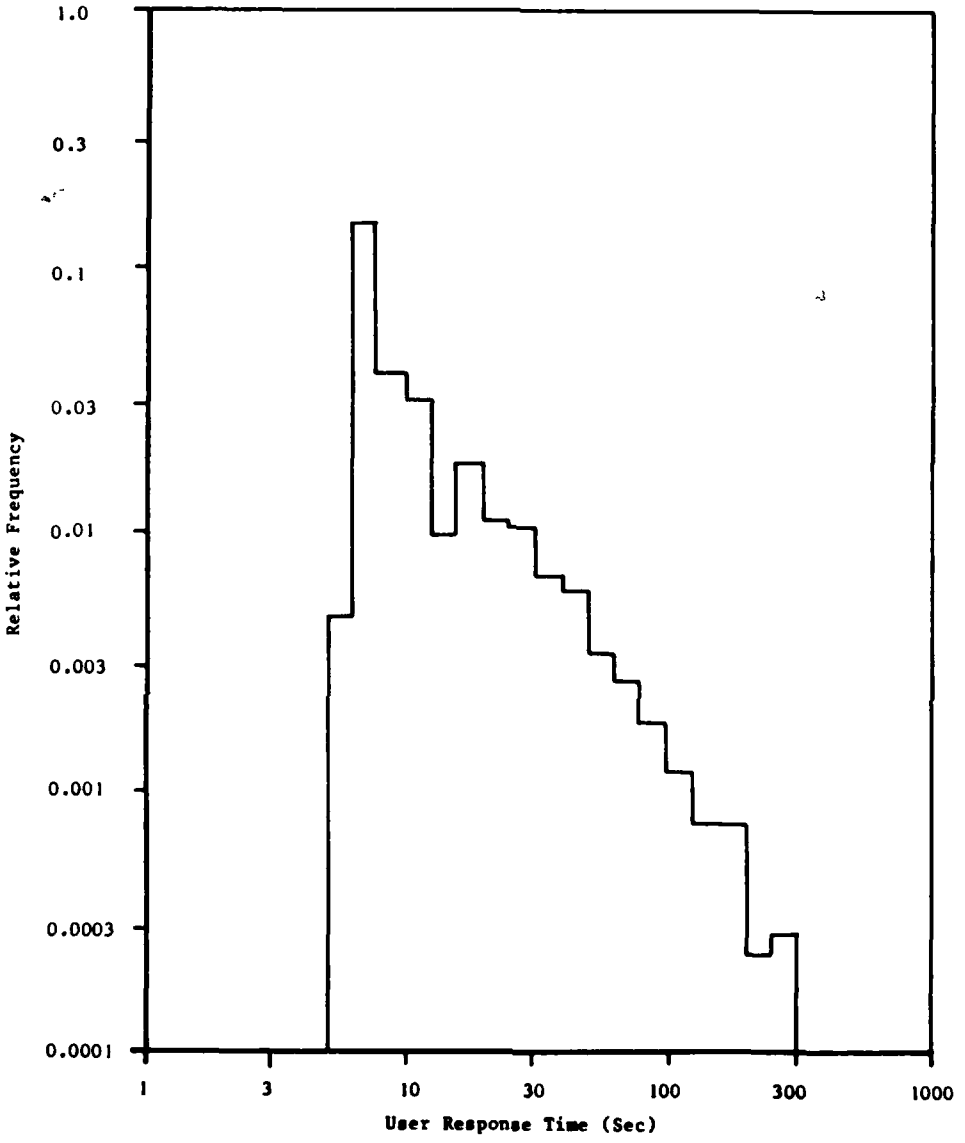


Figure 2-4. User Response Times

Since the index can become quite large, a sequential search is impractical. Consider for example a document retrieval system of five million documents, each having 70 words or tokens (see Table 2-2). The 350 million tokens contained in this document set would require 2.5 billion bytes (2.5 G-bytes) of storage, assuming an average of 7 characters per token. In an earlier project [Contract F30602-78-C-0065, reported in RADC-TR-79-313], the relationship of Equation 2-1 was found between the number of text tokens, T, and the number of vocabulary entries, V.

$$V = 1.41\ T^{0.85} \qquad (2\text{-}1)$$

Then, the 350 million tokens have a vocabulary of about 26 million entries, implying at least 180 MBytes of storage. In addition, a list of location pointers is associated with each vocabulary entry. Allowing three bytes for the document numbers, and one byte each for paragraph, sentence, and word numbers requires 6 bytes per token in the documents, or 2.1 G-bytes. Thus, the index of about 2.3 G-bytes requires nearly the same amount of storage as the documents (2.5 G-bytes).

To appreciate the advantage of having the documents indexed, assume the data base is stored in a disk system capable of storing 0.5 MBytes per cylinder. Then, the example document texts require 4800 cylinders of storage. Assuming, an arbitrarily fast processor, and 300 milliseconds to scan each cylinder, directly scanning all documents at search time for a given token, or set of tokens, requires 24 minutes. The index, however, can be used with a binary search method. Only one track per cylinder need be checked to control searching. If the average random access (seek plus latency plus scan) is 50 milliseconds, then the 13 tracks which must be read, to locate the desired cylinder, require 650 milliseconds. Adding 300 milliseconds to scan the final cylinder yields less than one second per search term to obtain the necessary document set pointers.

A Sample Set of Documents

```
DOCUMENT #1
06 HIGH ACCURACY WEAPONS
11 FRED SMITH
43 THE PRECISION OF REMOTELY GUIDED AND SMART MISSILES IS RAPIDLY
   IMPROVING.  SHIPS AND TANKS ARE ESPECIALLY VULNERABLE TO SUCH WEAPONS.

DOCUMENT #2
06 RIBOSOMES
11 ARTHUR JONES
43 RIBOSOMES SYNTHESIZE PROTEINS IN CELLS.  A RIBOSOME MODEL HAS BEEN
   DEVELOPED TO DESCRIBE THE STEPS OF THIS SYNTHESIS PROCESS.

DOCUMENT #3
06 MONSOONS
11 JAMES EDWARDS
43 SEASONAL WINDS, CALLED MONSOONS, SUPPLY WATER TO THE HALF OF THE
   EARTH'S POPULATION LIVING IN SOUTHERN ASIA.  COMPUTER SIMULATIONS ARE
   BEING DEVELOPED TO PREDICT MONSOONS.
```

Figure 2-5. The Major Components of a Data Base

The Corresponding Index

VOCABULARY	LISTS OF OCCURRENCE LOCATIONS*
A	02 43 02 01
ACCURACY	01 06 01 02
AND	01 43 01 06, 01 43 02 02
ARE	01 43 02 04, 03 43 02 03
ARTHUR	02 11 01 01
ASIA	03 43 01 17
BEEN	02 43 02 05
BEING	03 43 02 04
CALLED	03 43 01 03
CELLS	02 43 01 05
COMPUTER	03 43 02 01
DESCRIBE	02 43 02 08
DEVELOPED	02 43 02 06, 03 43 02 05
EARTH'S	03 43 01 12
EDWARDS	03 11 01 02
ESPECIALLY	01 43 02 05
FRED	01 11 01 01
GUIDED	01 43 01 05
HALF	03 43 01 09
HAS	02 43 02 04
HIGH	01 06 01 01
IN	02 43 01 04, 03 43 01 15
IMPROVING	01 43 01 11
IS	01 43 01 09
JAMES	03 11 01 01
.	
.	
.	
SYNTHESIZE	02 43 02 01
TANKS	01 43 02 03
THE	01 43 01 01, 02 43 02 09, 03 43 01 08, 03 43 01 11
THIS	02 43 02 12
TO	01 43 02 07, 02 43 02 07, 03 43 01 07, 03 43 02 06
VULNERABLE	01 43 01 06
WATER	03 43 01 06
WEAPONS	01 06 01 03, 01 43 02 09
WINDS	03 43 01 02

*Document, Paragraph, Sentence, and Word Numbers

Figure 2-5. The Major Components of a Data Base (Cont'd)

Two techniques are used to reduce the index lookup time. One technique is partitioning or segmentation, and the other is the segregation of the vocabulary from the occurrence lists (location pointers). Segmentation employs a table of selected vocabulary terms. Each table entry consists of a token (24 bytes in a fixed format) and a cylinder address (2 bytes). Thus, storing the first token for each cylinder of the example case requires less than 128 K Bytes if the resulting table is stored in main memory, only 300 milliseconds is required to find the desired entry on the target cylinder. The second technique, segregation, provides further improvement in performance.

The 300 milliseconds per cylinder can be reduced by segregating the vocabulary from the occurrence lists or pointers. Recalling that the total vocabulary has 26 million entries, storing this plus 26 million pointers to their associated occurrence lists requires on the order of 250 M Bytes of storage, or about 500 cylinders. Then, the same segment table of 5000 entries would locate the vocabulary word to a single track, and the vocabulary entry would locate its occurrence list to one track. Thus, the total lookup time is reduced to two random accesses at about 50 milliseconds each adding up to about 100 milliseconds.

The final data base organization is illustrated in Figure 2-6. With some simplifying assumptions, the preceding discussion can be generalized into a set of formulas giving the magnitudes of the data base elements. Let S_d denote the storage required for the texts of the documents. Then

$$S_d = T \cdot C_t \text{ bytes} \qquad (2\text{-}2)$$

where T is the number of tokens in the tests and C_t is the number of characters per item. Similarly, if S_v denotes the storage for the vocabulary, neglecting pointers to occurrence lists, then

$$S_v = V \cdot C_t \text{ bytes} \qquad (2\text{-}3)$$

where V is the number of vocabulary entries (see Equation 2-1). The size (S_o) of the occurrence lists is given by

$$S_o = T \cdot \log_{256} T \text{ bytes} \tag{2-4}$$

The base 256 logarithm accounts for the length in bytes of the location or occurrence pointers. The number of entries, N_s, in the segment table is

$$N_s = S_v/K \text{ entries} \tag{2-5}$$

and the size, S_s, of the table is on the order of

$$S_s = N_s \cdot C_v \text{ bytes} \tag{2-6}$$

where K is the capacity of one disk track (or other storage unit) and Cv is the number of bytes per entry. Figure 2-7 shows a graph of these formulas for $C_t = 7$, $C_v = 24$, and $K = 25000$. The respective values from the earlier detailed analysis are also plotted for comparison. Although the formulas seem to underestimate the storage requirements, the following consideration results in the formula being adequate.

The major consideration is that not all words in the documents are indexed. Common words such as conjunctions, articles, some prepositions and pronouns are eliminated from the vocabulary. These common words are also called stopwords. While the number of words eliminated is negligible (often only one to three hundred), they account for a significant number of tokens. Eliminating these stopwords can reduce the size of storage for the occurrence lists by up to 30%.

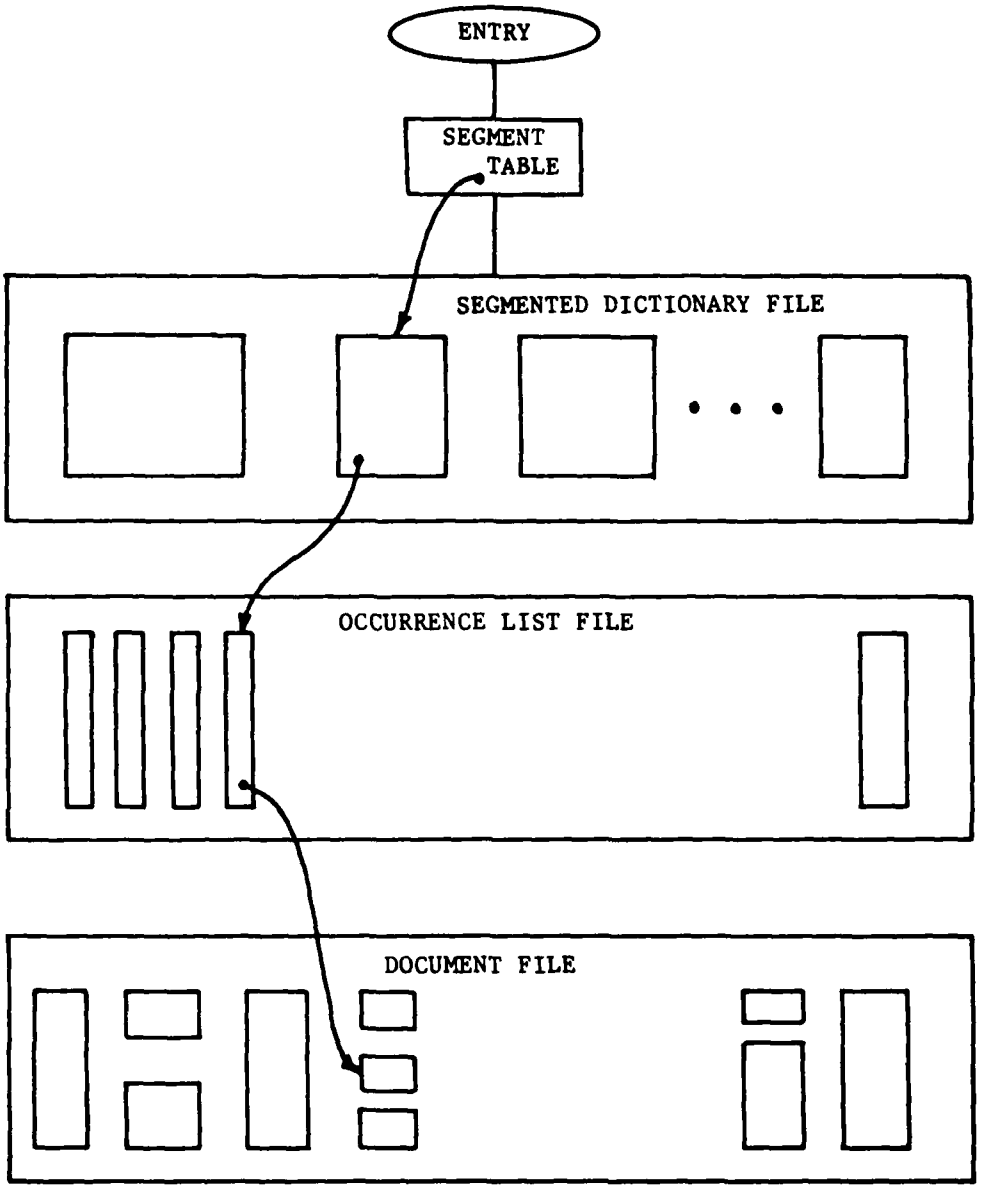


Figure 2-6. General Data Base Structures

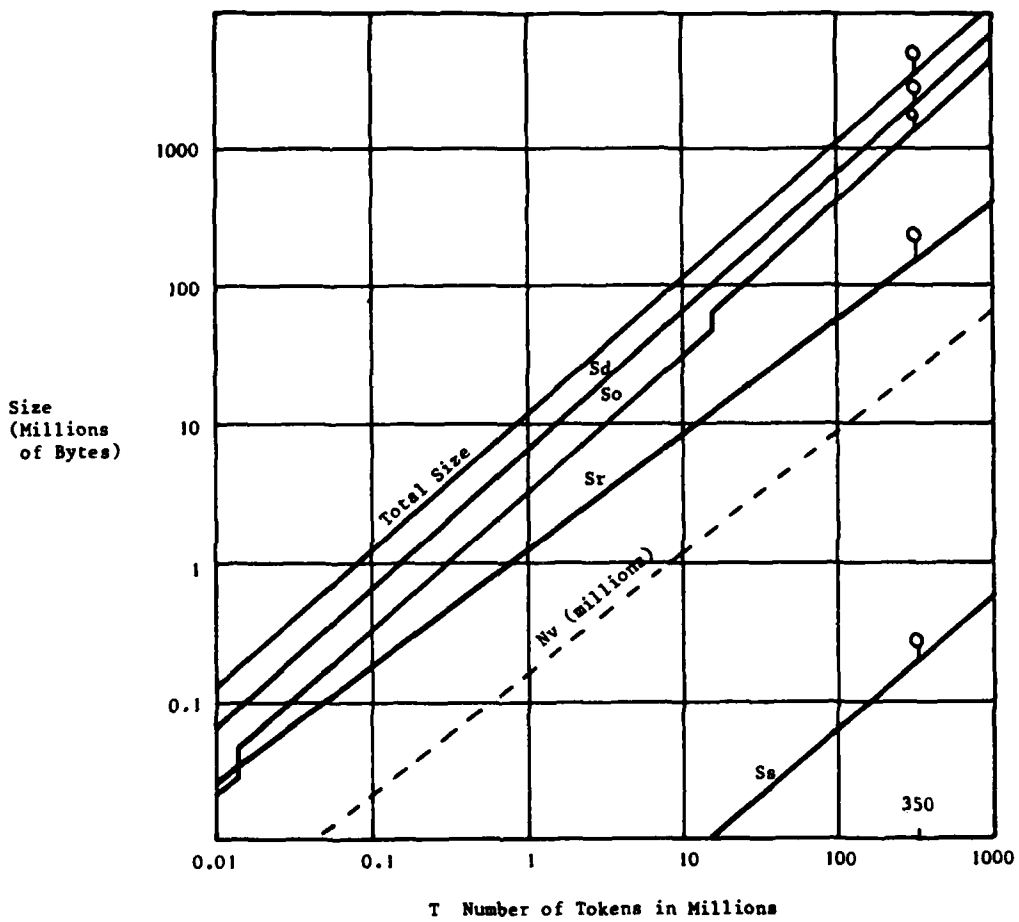


Figure 2-7. Sizes of Data Bases

Another characteristic of the data base needs to be mentioned. As documents or messages are received, the data base grows, and to meet fixed capacity limits, some documents in the data base must be deleted. This is readily supported by a multiplicity of data bases, each with the structure of Figure 2-6. If each data base covers a chronological period, then the oldest is often discarded to make room for new acquisitions. The data bases are catalogued and analysts may restrict their searching to a subset of them. The operation of merging two data bases into a single large one is useful in document retrieval and message handling systems. As can be seen, the total storage size for a system is effectively a linear function of the number of tokens in it. Thus, a system containing 100 million words of text in a single data base provides no essential advantage in storage over 100 data bases of 1 million tokens each. Some disadvantage however, will be experienced since searching all the small data bases for a token requires 200 disk accesses rather than only two accesses. The actual cost in time may not be much greater if the system is supported by a large number of independent disk units. Another cost in partitioning the data base in this way is experienced through additional software complexity. These costs must be traded off against the advantages in updating the data base as described in Section 6.

2.5 THE SEARCH PROCESS

The general nature of the searching of the document data base has been described in earlier sections. The objective of this section is to break down the process into a number of elementary processes for which computational and storage requirements can be readily estimated. To begin, consider the execution of a search command with the frequently used form 'word SEN word OR word' (Table 2-4) on the sample data set, the tokens 'AND', 'THE' and 'TO' will be used to form the expression 'AND' SEN 'THE' OR 'TO'. First, the search terms or operands must be looked up in the vocabulary, via the segment table, to locate their occurrence lists. Next the occurrence lists must be retrieved. The occurrence lists for the example tokens are shown in Figure 2-8. In an actual system these particular words would probably be considered stopwords and would not be found in the vocabulary.

'AND'	'THE'	'TO'	TEMP 'THE' OR 'TO'	REF #1 'AND' SEN TEMP
01 43 01 06	01 43 01 01	01 43 02 07	01 43 01 01	01 43 01 **
01 43 02 02	02 43 02 09	02 43 02 07	01 43 02 07	01 43 02 **
	03 43 01 08	03 43 01 07	02 43 02 07	
	03 43 01 11	03 43 02 06	02 43 02 09	
			03 43 01 07	
			03 43 01 08	
			03 43 01 11	
			03 43 02 06	

Figure 2-8. Execution of a Sample Expression

The first operation performed is the OR of the lists for 'THE' and 'TO'. Because of the way the document, paragraph, sentence, and word numbers are assigned in the update process, the occurrence entries are listed in increasing numerical order. Thus, forming the union of these ordered sets can be done by a modified merge rather than comparing each entry of one list with each of the other. The result of this merge is also an ordered list as shown in Figure 2-8. Because no two words can occupy the same position in any document the list, called temp, for 'THE' OR 'TO' could have been formed by concatenating the lists for the two search tokens. This, however, would result in an unordered list which would make subsequent operations much more difficult.

Next, the SEN operation is executed using the list for 'AND' and the temp list. Again, a merge type process is used, but now the word number is ignored. In performing the merge of these two lists, the document, paragraph, and sentence numbers match in two cases, both in document #1. As shown in the example of Figure 2-8, the word numbers in the final result list are undefined. Since this result list is also an ordered list, it too can be used as an operand. (Conventions for dealing with undefined values are discussed later.) As a result of executing this search command, the analyst is informed by the system that two occurrences satisfy the expression and that these occurrences span one document.

Following this example, the search process is decomposed as shown in Figure 2-9. Of these tasks, the segment table lookup, the locating of tokens in the dictionary, and the execution of operations on occurrence lists were of primary interest in this project because of their apparent suitability for the processor architectures to be evaluated in this project.

The segment table lookup is a simple task in which the pointer to the correct dictionary segment must be found by finding the lexographically largest entry (first word in the segment) which is smaller than or matches each token type search operand. From the analysis of Section 2.3, the required lookup rate is only 4.8 lookups per second.

The process of finding the tokens in the dictionary segments is only slightly more complex if the dictionary is packed rather than using a large fixed field as is done in the segment table. Assume that the segments are packed and the format for each entry is a field giving the length (number of characters) of the token, followed by a variable length field containing the token, which is followed by a field for the address of the token's occurrence list. Finding the search token in the segment is easily approached by considering the segment as a constant rate character stream. First, the length of an entry must be tested. If a truncated token was specified, then the length must be tested against an upper and lower limit. If these tests succeed, then each character of the entry (up to the minimum limit) must be tested against the corresponding character of the search token. If these tests also succeed, then the occurrence list address of the token must be captured and stored. Once a match has been found, or a non-match following a match when a truncation is specified, then the segment character stream can be discontinued or ignored. Since the segment is scanned sequentially, half a segment must be tested, on the average, for each search token. The computational effort is about two byte-length comparisons per character. If the length of a segment is Kbytes (see Equation 2-5) then the effort per search token is K comparisons ($1/2 \times 2 \times K$).

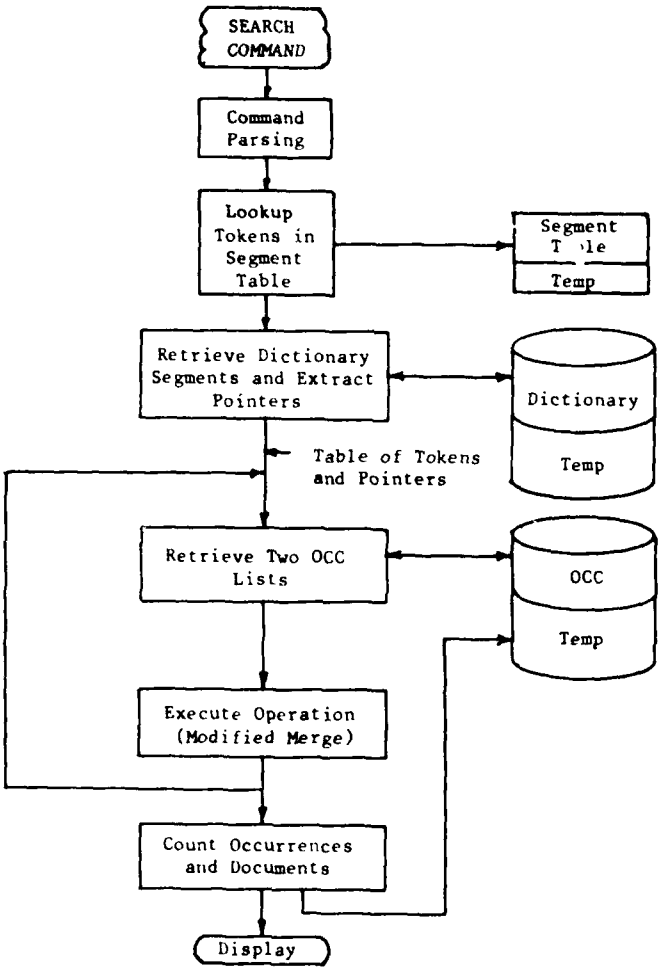


Figure 2-9. Search Process

The next operation of interest is the execution of the modified merges to implement the search operations. Only the three most frequently used operators (OR, SEN, and ADJ), which account for 96% of the operations used, are considered here. All of the operators (except NOT) use the same basic process. First, the first occurrence pointer is taken from each of the two lists. They are compared and then some action is taken while discarding one or both pointers. Then, each discarded pointer is replaced by the next pointer from the appropriate list. The comparison, action, and replacement are repeated until one list is exhausted, at which time only one action is executed over the residue of the remaining list. (NOT operations are implemented through modified actions during other operations.)

First, consider the OR operation. Although it is specified 80% of the time, it must be executed more than that since it is used to combine multiple lists retrieved for a truncated term. The comparisons and actions performed for the OR operator depend on the presence of undefined pointer fields resulting from previous operations (such as PAR). Assume that each list is tagged to indicate which fields of the pointers are undefined. For example, let the expression (a, b, c, d) indicate that all fields of the left operand's list are specified, and let (a', b', *, *) show that the sentence numbers and word numbers are undefined in the right operand's list (such as from a PAR operation). The operation performed is governed by the least specified list. In this case, the comparison (a,b) : (a', b') is made if they match, then (a, b, *, *) is added to the result list. If (a,b) is less than (a', b') then, (a, b, *, *) is added to the result, and a new pointer is taken from the left list. The third possibility results in an output and replacement of (a', b', *, *).

This example is repeated in Table 2-6 with the definition of modified merges for other operator cases.

TABLE 2-6. PROCESSING OF MODIFIED MERGES

Operation	Least Specified Operand Pointers	Comparison	Output to Result List	Operand(s) Replaced
OR	(a,b,c,d)	(a,b,c,d) < (a',b',c',d') (a,b,c,d) > (a',b',c',d')	(a,b,c,d) (a',b',c',d')	(a,b,c,d) (a',b',c',d')
	(a,b,c,*)	(a,b,c) < (a',b',c') (a,b,c) > (a',b',c') (a,b,c) = (a',b',c')	(a,b,c,*) (a',b',c',*) (a,b,c,*)	(a,b,c,*) (a',b',c',*) both
	(a,b,*,*)	(a,b) < (a',b') (a,b) > (a',b') (a,b) = (a',b')	(a,b,*,*) (a',b'*,*) (a,b,*,*)	(a,b,*,*) (a',b',*,*) both
SEN	(a,b,c,d)	(a,b,c) < (a',b',c') (a,b,c) > (a',b',c) (a,b,c) = (a',b',c')	none none (a,b,c,*)	(a,b,c,d) (a',b',c',d') both
	(a,b,c,*)	(a,b,c) < (a',b',c') (a,b,c) > (a',b',c') (a,b,c) = (a',b',c')	none none (a,b,c,*)	(a,b,c,*) (a',b',c',*) both
	(a,b,*,*)	command rejected	as	invalid
ADJ	(a,b,c,d)	(a,b,c,d+1) < (a',b',c',d') (a,b,c,d+1) > (a',b',c',d') (a,b,c,d+1) = (a',b',c',d')	none none (a',b',c',d')	(a,b,c,d) (a',b',c',d') both
	(a,b,c,*)	command rejected	as	invalid
	(a,b,c,*)	command rejected	as	invalid

Except in cases where the command is rejected before execution because of incompatible operands, the operations require one 3-way test and branch per occurrence in both operand lists (assuming matches are infrequent). Since the OR operation is dominant (at least 80%) and most operands are words or truncations (70%), the first case of Table 2-6 is encountered most of the time and usually no matches will occur for it. Thus, the computational effort is essentially one comparison per occurrence pointer of both lists. In a previous project [Contract F 30602-78-C-0065] some information on occurrence list length for search tokens was obtained. The distribution of lengths is repeated here as Figure 2-10. In that project, the average number of occurrences per search token used was found to be 2905 occurrences, even though 64% of the search tokens had fewer than 70 occurrences. Based on this average, 5800 comparisons are required per search operation on the average, and with reference to Table 2-5, an average of 1450 comparisons per second per user are required.

The computational requirements for those tasks of the search process analyzed in this section are tabulated in Table 2-7.

TABLE 2-7. COMPUTATIONAL REQUIREMENTS OF SEARCHING PER ACTIVE USER

Task	Requirement
Segment Table Lookupp	0.24 lookup operations per second
Dictionary Segment Scan	0.24xK byte comparisons per second
Search Operation (Modified Merge)	1450 occurrence pointer comparisons per second

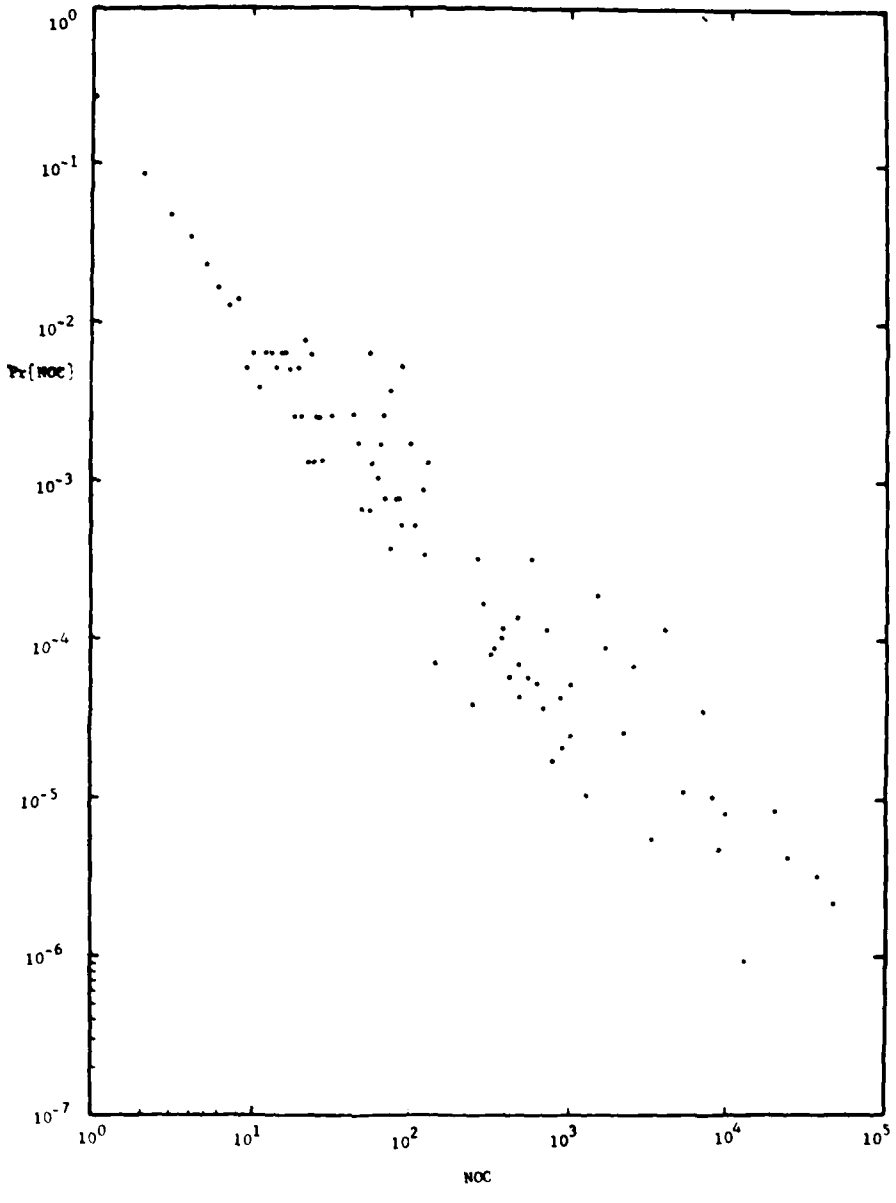


Figure 2-10. PDF of Number of Occurrences of Search Tokens

2.6 THE UPDATE PROCESS

The update process is executed whenever new documents are to be added to the system, and consists of two basic steps (Figure 2-11). First, a temporary data base which is compatible with the main data base is created for a batch of new documents. Then, this temporary data base is catalogued or merged with an existing data base to make it available for searching. In message handling applications, it may be necessary to update the system as each message is received. Since a data base cannot be searched during a data base merge, and the merge time increases with data base size, it may be necessary to manage two copies of a buffering data base between these steps. Nevertheless, the same basic tasks are required for both document retrieval and message handling applications.

The step which creates a data base, given a set of documents or messages, can be viewed as a sequence of four tasks. The lexical decomposition task breaks the character string comprising each document into words or lexical units and assigns to each its document, paragraph, sentence, and word number. The algorithm used for lexical decomposition depends on the syntax assumed for the text. For example, the computational requirements depend on whether it is necessary to distinguish the use of periods in numbers, abbreviations, and at the ends of sentences. In this project, that distinction, though important in a production system, was not made in order to reduce the programming effort. Instead, the simple document syntax of Figure 2-2 was assumed. For this syntax, it was sufficient to determine for each input character to which of the three sets it belonged. One set consists of those characters which can be part of a token and includes the letters, numbers, the hyphen, the slack, and the apostrophe. The second set contains only the period, which signals the end of a sentence, and the third set contains all other characters, which are considered to be equivalent to the blank. A simple sequential machine which is adequate for finding sentences and tokens (but not

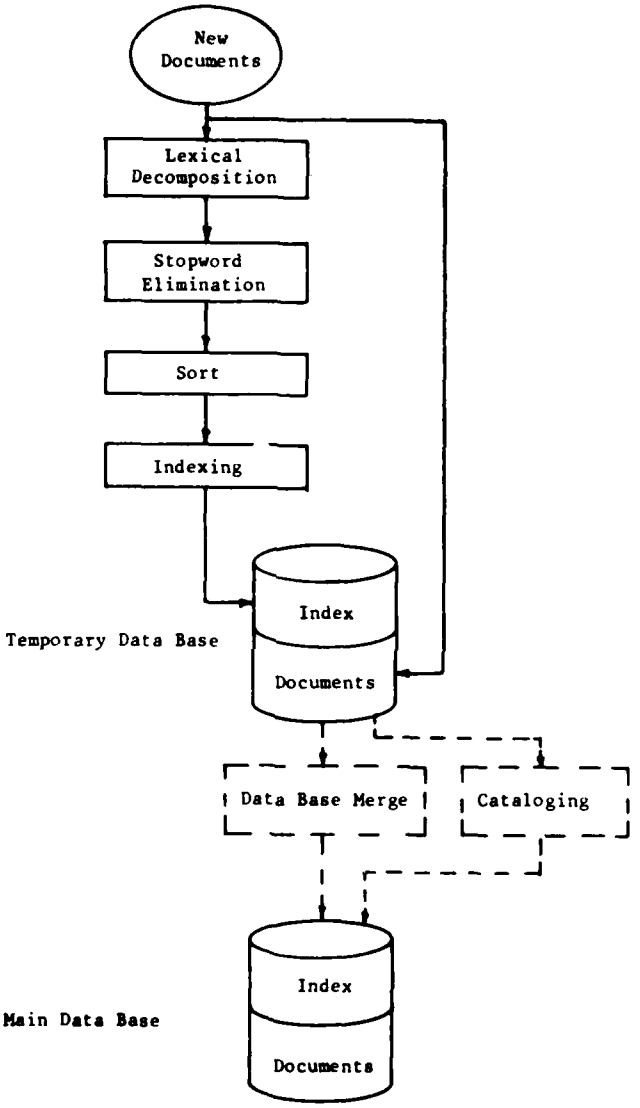


Figure 2-11. The Update Process

Current State	Current Input Character	Next State	Action(s)
WAIT	<blank>	WAIT	None
	<letter>	TOKEN	Put input into token accumulator
	<period>	WAIT	Increment sentence number. Set word number to 1.
TOKEN	<blank>	WAIT	Output token accumulator with occurrence pointer. Increment word number.
	<letter>	TOKEN	Add to token accumulator.
	<period>	WAIT	Output token accumulator with occurrence pointer. Increment sentence number. Set word number to 1.

Figure 2-12. A Sequential Machine for Lexical Decomposition

paragraph and document boundaries) is shown in Figure 2-12. Because the character set is small (256 entries) and the machine states are few, the state-character combination could be used to address a memory of 512 registers. Each register needs one bit to specify the next state part of the next address, and three to six bits to specify the actions to be taken. The actions themselves consist of increment and store type operations. Since the machine should be going from the token state to the token state or the wait state to the wait state most of the time, the computational effort per input character should be about 3 operations, two of which are a memory read and a memory write. Based on the estimate of 7 characters per token, the lexical decomposition effort is 21 operations per token.

The next task is the elimination of common or stopwords. This is of the same complexity as the segment table lookup. Although stopword elimination could be done more efficiently by a merge following the sort, it is done now because it eliminates a significant number (20% to 30%) of the tokens that have to be sorted. The computational effort then, for stopword elimination, is one lookup per document token.

The next task is to sort the tokens (with their occurrence pointers) into lexicographical order. Although many sorting techniques have been developed, it will be assumed that a merge-sort method is used here. The merge-sort uses an algorithm which is nearly the same as the OR search operation (Section 2.5). The computational effort for this task is $T \log_2 T$ comparisons where T is the number of tokens in the documents.

The final task involves the formation of the dictionary and occurrence lists. This is done by comparing each entry of the sorted list of tokens against a reference. The reference is initially set to the first token, and its occurrence pointer is put into an occurrence list. As a new token is taken from the sorted list, it is compared to the reference. If

it matches, its pointer is added to the occurrence list being formed. If it does not match, the current occurrence list is filed and its address is appended to the reference, which is then filed in the dictionary. The current token then becomes the reference, and its pointer starts a new occurrence list. Although seemingly complex, the computational effort of the indexing task is about one comparison per document token. If the document set to be added to the data base contains fewer than 1000 tokens, the comparison will show no match most of the time and the relative computational effort will be greater.

The second step of the update process may require a data base merge. This involves the combination of two document files, two dictionaries, and two occurrence list files. Assuming the document numbers assigned to the new documents began in sequence after the highest document number in the data base being updated, it is only necessary to append the new documents onto the old file. If this is not the case, the document numbers in the new document file and new index files must be modified by adding a fixed constant.

Dealing with the index involves a merge type of operation which is governed by the dictionary files. Two dictionary entries are taken and compared. The lesser token (lexicographically) is added to the updated dictionary, with the current address of the updated occurrence files. The occurrence list of the lesser token is placed at that address, and the address is changed to the next position beyond the last occurrence pointer's address. The lesser dictionary token is then replaced by the next entry from the same dictionary. In the event the dictionary tokens match, then one entry is made in the updated dictionary, the occurrence list for the main data base is output followed by the list for the update or temporary data base, and both dictionary tokens are replaced from their respective files. As with the other merge type operations, the computational load is essentially one comparison per vocabulary entry, or $1.41T^{0.85}$ comparisons, using Equation 2-1.

At the beginning of this chapter, the task of forming mailing lists for new documents, based on mission profiles, was associated with the update process. As indicated in Section 2.2, the mission profiles are largely composed of search commands. Thus, forming the mailing list can be done by executing the profile file on the temporary data base for the new documents.

This approach to routing new documents is adequate for document retrieval systems which are updated periodically by large document sets. For message handling systems, however, which must quickly route messages received at random intervals, an alternative approach is indicated. An attractive alternative employs a dictionary of profile tokens, which can be merged with the dictionary of a message. Any common terms can be used to pick off the occurrence lists of interest. Pairs of pointers between profile tokens and the occurrence lists can then be sorted based on the profile statement numbers. Following this, the profile statements can be executed.

The computational requirements of the update process are given in Table 2-8. The requirements of the search operations depend on the average length of the occurrence lists for the search tokens, with respect to the temporary data base. It is assumed here that this average depends linearly on the number of tokens in the data base. This is a reasonable assumption based on occurrence list length growth rates determined in an earlier study. The average length, L, for a base of T tokens is

$$L = \frac{2905}{350 \times 10^6} \quad T = 8.3 \times 10^{-6} \quad T \text{ occurrences} \qquad (2\text{-}7)$$

TABLE 2-8. COMPUTATIONAL REQUIREMENTS OF THE UPDATE PROCESS

Process and Tasks	Requirements
Temporary Data Base Formation	
Lexical Decomposition	21T basic operations (1-byte operand)
Stopword Elimination	T lookups (12-byte operands)
Sort	T $\log_2$ T compares (24-byte operands)
Index Formation	T compares (24-byte operands)
Data Base Merge	$1.41T^{0.58}$ compares (24-byte operands)
Profiling Search	
Segment Table	3.5C lookups (24-byte operands)
Segment Scan	3.5C K character-compares (21-byte operands)
Search Operation	4×10^{-6}C T compares (6-byte operands)

NOTE: T tokens in the documents to be added
C profile search commands
K bytes per dictionary segment

2.7 ALLOCATION OF TASKS

As mentioned earlier, three types of processors were available to use in the demonstration: a general-purpose minicomputer (PDP11/70), a very high-performance integer array processor (AFP), and an associative unit (AU). The characteristics of the latter two machines are described in Section Three.

For demonstration purposes, it was decided to allocate tasks to the various equipment as shown in Table 2-9. Other control and data manipulation tasks which were performed in the PDP-11/70 are not listed in this allocation, but supported operation of the AU and AFP's.

TABLE 2-9. TASK ALLOCATION

Task	Machine
Search	
Segment Table Lookup	AU
Segment Scanning	AFP
Occurrences Merge	AFP
Update	
Lexical Decomposition	AFP
Stopword Elimination	AU
Sort	AFP
Indexing	AFP

2.8 RELATED TOPICS

This section covers topics or enhancements which were considered during the work of the project but are not essential to document retrieval and message handling.

2.8.1 Word Compression

The purpose of the word compression step is to reduce the number of bits required to store the tokens. The method used here offers about a 2 to 1 compression ratio. This doubles the rate at which comparisons, for sorting etc., can be made and halves the storage requirements for the vocabulary and the text. It should be noted that this step is not essential in document retrieval but is employed here for performance enhancement. There is an element of cost in compressing the dictionary entries with respect to the search process.

The text compression scheme used here is based on fully utilizing the 256 codes representable in an 8 bit byte. This is done using a table of 1-,2-,3-, and 4- letter patterns (called n-grams). There are up to 256 n-grams in the table, and one of the 256 possible 8-bit codes is associated with each. An example of such a table appears as Table 2-10 and some possible encoding of the token "perception" with this table is shown in Table 2-11. Although there may be many encodings of a particular token, all encodings decode to the original token by using the same table.

The encoding algorithm used in the current project processes each token from left to right. The longest n-gram matching the left of the token is found in the table, and its code is returned. The encoded characters are deleted from the token and the process is repeated until the token has been consumed. With this algorithm, the first encoding of "perception" is found, though the shortest encoding may not be found for all tokens.

The compression ratio obtained with this method depends on both the text to be compressed and the n-grams constituting the table. No optimal set of n-grams is known, nor is an efficient algorithm known for finding an optimal set for a given corpus of text. (Theoretically, the set of

TABLE 2-10. AN n-GRAM ENCODING TABLE

A	001	CONT	039	GY	077	M	115	PART	153	TE	191
AC	002	CR	040	H	078	MA	116	PE	154	TED	192
ACT	003	CT	041	HA	079	MAN	117	PER	155	TER	193
AD	004	CUL	042	HAS	080	MAR	118	PL	156	TH	194
AGE	005	D	043	HAT	081	MAT	119	PLA	157	THAT	195
AL	006	DA	044	HE	082	ME	120	PO	158	THE	196
ALS	007	DI	045	HER	083	MENT	121	PRE	159	THEI	197
AM	008	DU	046	HI	084	MI	122	PRES	160	THI	198
AN	009	E	047	HO	085	MIN	123	PRO	161	TI	199
AND	010	EA	048	HORM	086	MO	124	Q	162	TION	200
APP	011	EAR	049	I	087	MP	125	R	163	TO	201
AR	012	EC	050	IA	088	MPL	126	RA	164	TR	202
ARE	013	ED	051	IC	089	N	127	RE	165	RS	203
ARI	014	EL	052	ID	090	NA	128	REA	166	TU	204
AS	015	EM	053	IES	091	ND	129	RES	167	TURE	205
AT	016	EME	054	IG	092	NE	130	RG	168	TY	206
ATE	017	EN	055	IL	093	NG	131	RI	169	U	207
ATIO	018	ENE	056	IM	094	NI	132	RO	170	UC	208
ATOR	019	ENT	057	IN	095	NN	133	RS	171	UL	209
B	020	ER	058	ING	096	NO	134	RT	172	UM	210
BE	021	ERI	059	INS	097	NOT	135	RU	173	UN	211
BET	022	ES	060	INT	098	NS	136	RY	174	UR	212
BI	023	ET	061	ION	099	NT	137	S	175	URE	213
BLE	024	EV	062	IR	100	O	138	SE	176	US	214
BO	025	EVE	063	IS	101	OC	139	SH	177	UT	215
BUT	026	EX	064	IT	102	OD	140	SI	178	V	216
BY	027	EXP	065	ITS	103	OF	141	SM	179	VA	217
C	028	F	066	IV	104	OL	142	SMAL	180	VE	218
CA	029	FA	067	J	105	OM	143	SO	181	VER	219
CAL	030	FI	068	JE	106	ON	144	SP	182	VI	220
CAN	031	FO	069	K	107	ONE	145	SS	183	W	221
CE	032	FOR	070	KE	108	OP	146	ST	184	WA	222
CENT	033	FORM	071	L	109	OR	147	STA	185	WE	223
CES	034	FR	072	LE	110	OS	148	STE	186	WI	224
CH	035	G	073	LI	111	OT	149	SU	187	WITH	225
CL	036	GES	074	LL	112	OU	150	SYST	188	X	226
CO	037	GH	075	LO	113	P	151	T	189	Y	227
CON	038	GR	076	LY	114	PA	152	TA	190	Z	228

TABLE 2-10. AN n-GRAM ENCODING TABLE (CONT'D)

ZE	229
0	230
1	231
2	232
3	233
4	234
5	235
6	236
7	236
8	238
9	239
+	240
-	241
*	242
/	243
(	244
)	245
$	246
=	247
,	248
.	249
=	250
	251
	252

TABLE 2-11. SOME ENCODINGS OF "PERCEPTION" USING TABLE 2-10

	ENCODINGS	COMPRESSION RATIO
1	155,032,151,200	2.5
2	151,058,028,047,151,200	1.67
3	154,163,032,151,199,144	1.67
4	151,058,032,151,189,099	1.67
5	154,163,032,151,200	2
6	155,032,151,199,144	2
7	151,047,163,028,047,151,189,087,138,127	1

n-grams should be used equally frequently in encoding.) In the present system the field length for the token value is fixed, so compression is more important on long tokens than on short ones. Therefore, in a production system, the n-grams should be selected to minimize the field length rather than maximize the compression ratio.

As with lexical decomposition, the computational effort for word compression is proportional to the number of characters in the input documents.

For this project, word compression was allocated to the associative unit and was inserted between the tasks of lexical decomposition and sorting (See Figure 2-11). As mentioned earlier, word compression complicates the search process. The problem arises with truncated tokens as search operands. Because multiple characters can be represented in one code, it

is quite possible that a truncation is essentially specified in the middle of an n-gram. This can be handled by retrieving all dictionary entries satisfying a truncation to the next lower n-gram. Then, the problematic n-gram can be decoded for each entry and the spurious entries eliminated.

Since the size, Sd, of the document texts dominates the data base storage requirements (Figure 2-7), it is well worth compressing the document file, which can be done without compressing the dictionary and complicating searches. The decision to compress the dictionary must be made in view of the specific application. Having an associative unit available can save time in updating by reducing the size of comparison operations required in sorting and indexing.

2.8.2 Spelling and Morphology

System performance depends significantly on the quality of the input documents. Since every token is indexed, misspellings are indexed. Figure 2-13 shows a fraction of the FTD dictionary. Associated with each token is the length of its occurrence list in the data base. This sample contains 155 tokens, of which 62 (or 40%) are misspellings. Thus, nearly half of this dictionary could be eliminated by a spelling checking and correcting program. Possibly, more significant than this is that these misspellings represent 132 occurrences, and that some vital information might not be retrieved in a search because these occurrences would be missed.

A second consideration involves suffixes and compounds (formed with hyphens). Table 2-12 shows the suffix and compounding morphology of the sample dictionary tokens. Since the truncation form of search terms is most often used to eliminate distinctions based on suffixes and compounds, it is well to consider eliminating such distinctions from the

17,774	DIRECT
4	DIRECT-
1	DIRECT-ACTING
11	DIRECT-ACTION
2	DIRECT-CHANNEL
1	DIRECT-CHARGE
1	DIRECT-CHILL
2	DIRECT-CONVERSION
5	DIRECT-COUPLED
1	DIRECT-COUPLING
1	DIRECT-CURRENT
127	DIRECT-CURRENT
1	DIRECT-DIESELECTRIC
2	DIRECT-DRIVE
49	DIRECT-FLOW
1	DIRECT-HEAT
1	DIRECT-HEATING
1	DIRECT-HIT
1	DIRECT-ILLUMINATION
4	DIRECT-INDIRECT
4	DIRECT-INJECTION
4	DIRECT-INVERSE
1	DIRECT-ION
5	DIRECT-LINE
4	DIRECT-LINEAR
1	DIRECT-OF
1	DIRECT-POLARITY
1	DIRECT-RADIATING
6	DIRECT-READING
1	DIRECT-RECIRCULATON
1	DIRECT-RUN
1	DIRECT-SHADOW
1	DIRECT-STROKE
1	DIRECT-THERMOMETRIC
1	DIRECT-VIEW/SCAN-CONVERTER
1	DIRECT-VISIBILITY
2	DIRECT-VOLTAGE
1	DIRECT/STRAIGHT
6	DIRECTABILITY
2	DIRECTABLE
2	DIRECTACTING
1	DIRECTCURRENT
1	DIRECTEDNESS
1	DIRECTDNESS
4819	DIRECTED
2	DIRECTED-ACTION
1	DIRECTEDALONG
5	DIRECTEDNESS
1	DIRECTEDNROMAL
1	DIRECTEDTED
1	DIRECTEDTOWARDS
1	DIRECTELEKTRIC
1	DIRECTELY
3	DIRECTER
2	DIRECTERMOM
6	DIRECTEUR
1	DIRECTFLOW
1	DIRECTHERM
5	DIRECTHERMOM
1	DIRECTHERMON
1	DIRECTIA
3	DIRECTII
1	DIRECTIM
4	DIRECTIN
4	DIRECTINAL
808	DIRECTING
1	DIRECTINOS
1	DIRECTINTEGRATION
12	DIRECTIO
1	DIRECTIOANL
16095	DIRECTION
1	DIRECTION·THE
1	DIRECTION(OIO)
1	DIRECTION-
1	DIRECTION--THE
1	DIRECTION--WITH
1	DIRECTION-A
3	DIRECTION-AND-RANGE
1	DIRECTION-CHANGING
1	DIRECTION-DEPENDENT
2	DIRECTION-FINDING
1	DIRECTION-RESEARCH
1	DIRECTION-SELECTING
2	DIRECTION-SELECTIVE
2	DIRECTION-SENSOR
1	DIRECTION/DISTANCE
2	DIRECTIONA
2	DIRECTIONABILITY
1729	DIRECTIONAL
2	DIRECTIONAL-CRYSTALLIZATION
48	DIRECTIONALITY
63	DIRECTIONALLY
9	DIRECTIONALNESS
4	DIRECTIONALS
1	DIRECTIONALZONALITY
2	DIRECTIONAND
1	DIRECTIONASOLIDIFICATION
1	DIRECTIONED
4	DIRECTIONER
2	DIRECTIONES
2	DIRECTIONFINDER
1	DIRECTIONFOR
5	DIRECTIONING
2	DIRECTIONLESS

Figure 2-13. A Sample of a Dictionary

2	DIRECTIONN
6	DIRECTIONOF
1	DIRECTIONOVER
5665	DIRECTIONS
1	DIRECTIONS(111
1	DIRECTIONS-
1	DIRECTIONS--WITH
1	DIRECTIONS--FROM
1	DIRECTIONS--IN
2	DIRECTIONS--LENGTHWISE
1	DIRECTIONS-OPERATIVE
1	DIRECTIONSD
2	DIRECTIONSIN
1	DIRECTIONWERE
1	DIRECTIOR
1	DIRECTIORS
1	DIRECTIOS
2	DIRECTIOSN
1	DIRECTIOV
1	DIRECTIV
478	DIRECTIVE
2	DIRECTIVEITY
1	DIRECTIVELY
4	DIRECTIVENESS
570	DIRECTIVES
2	DIRECTIVITIES
290	DIRECTIVITY
1	DIRECTIX
1	DIRECTL
2	DIRECTLAYING
1	DIRECTLED
1	DIRECTLEY
1	DIRECTLINE
1	DIRECTLIR
7796	DIRECTLY
2	DIRECTLY-DISTILLED
1	DIRECTLY-GROUNDED
1	DIRECTLY-LINEAR-CONTROLLED
2	DIRECTLYON
1	DIRECTMEMORY
6	DIRECTNESS
5	DIRECTO
6	DIRECTON
1	DIRECTIONAL
1	DIRECTIONATE
5	DIRECTONS
26,923	DIRECTOR
5	DIRECTOR-
1	DIRECTOR-PROF
9	DIRECTOR-GENERAL
2	DIRECTOR-GENERALS
2	DIRECTOR-REPRESENTATIVES
1	DIRECTOR-TO-SECRETARY
1	DIRECTOR/AUTOMATIC
11	DIRECTOR'S
1	DIRECTORAAT
1	DIRECTORAL
574	DIRECTORATE
2	DIRECTORATE-GENERAL
1	DIRECTORATE'S
1	DIRECTORATEFO
89	DIRECTORATES
2	DIRECTORIA
2	DIRECTORIA-GERAL
3	DIRECTORIAL
1	DIRECTORIE
15	DIRECTORIES
878	DIRECTORS
3	DIRECTORS'
12	DIRECTORSCOPE
20	DIRECTORSHIP
127	DIRECTORY
1	DIRECTORY-SECRETARY
2	DIRECTOS
1	DIRECTRECORDING
1	DIRECTRED
1	DIRECTREX
12	DIRECTRICES
22	DIRECTRIX
1	DIRECTRIXES
2	DIRECTRO
1	DIRECTRY
251	DIRECTS
1	DIRECTTORS
1	DIRECTU
1	DIRECTURATE
1	DIRECTY
1	DIRECTYLY

Figure 2-13. A Sample of a Dictionary (Cont'd)

TABLE 2-12. MORPHOLOGY OF THE TOKENS

WORD (ENDING)	INDIVIDUAL ENTRIES WORDS	INDIVIDUAL ENTRIES TOKENS	TOTAL ENTRIES WORDS	TOTAL ENTRIES TOKENS
DIRECT	1	17,774	51	31,957
-	36	252		
ABILITY	1	6		
ABLE	1	2		
ED	1	4,819		
EDNESS	1	1		
ING	1	808		
IVE	1	78		
IVES	1	70		
IVITY	1	90		
LY	1	7,796		
LY-	3	4		
NESS	1	6		
S	1	251		
DIRECTION	1	6,095	24	13,626
-	12	17		
AL	1	1,729		
ALITY	1	48		
ALLY	1	63		
LESS	1	2		
S	1	5,665		
S-	6	7		
DIRECTOR	1	26,923	12	28,498
-	6	20		
'S	1	11		
ATE	1	574		
ATES	1	89		
S	1	878		
S'	1	3		
DIRECTORSHIP	1	20	1	20
DIRECTORY	1	127	3	143
-	1	1		
*IES	1	15		
DIRECTRIX	1	22	2	34
*CES	1	12		
MISSPELLINGS			62	132
TOTAL			155	74,410

dictionary. Table 2-12 shows that the 155 tokens in the sample could be covered by only six root words if misspellings were eliminated. This would imply a dictionary of only 4% of the current size. If, indeed, this could be done, it could be feasible to store the entire dictionary in semiconductor memory, eliminating half the disk accesses in searching. This modification requires a reliable suffix analyzer, versions of which are becoming available.

2.8.3 Cache Memory

This section deals with the possible advantages of including a large semiconductor memory in the system to act as a cache memory between the disk memory and the main memories of the processors. The primary purpose of such a cache is to minimize the time for disk accesses and data transfers with the disks.

In current practice, the pointers to occurrence lists which are the result of search operands may be stored in fast memory, but the lists, being large, are often stored on disk once they are formed. Since the likelihood that they will be referenced in a subsequent statement is high, search results are very good candidates for a cache. With occurrence list lengths of 3000 pointers and assuming 6 bytes per pointer, about 20,000 bytes are required per result.

Another method of reducing disk accesses involves placing a portion of the dictionary in a cache. To determine the cache requirements for a partial dictionary, the search terms used in the mission profiles were examined. Almost half the search tokens were used only once, and only 33% of the 30,000 search tokens were used more than twice. Storing each token when it is first used would reduce accesses for the dictionary by only a third. Since the dictionary involves only half the total accesses, the net savings would be only on the order of 10%. In most

cases, time would be added to search the cache. The storage required for 30,000 tokens at 24 bytes each is about 750,000 bytes. This would be an excellent application for the associative unit. Because the searches in the profiles have undoubtably been designed to compensate for the clutter in the dictionary (See Section 2.8.2), caching of frequently used tokens for searching should be reexamined after the dictionary has been cleaned up.

3.0 ARRAY AND ASSOCIATIVE PROCESSORS

The system configuration available to demonstrate the Advanced Document Retrieval System was configured from equipment at Control Data's Information Processing Center. This equipment consists of three Advanced Flexible Processors (AFPs), the Associative Memory Unit, a DEC PDP 11/70, and a mass storage subsystem consisting of 13 processors -- a total of 18 processors. The mass storage processors are CDC 7600 peripheral processing units.

3.1 ADVANCED FLEXIBLE PROCESSOR ARRAY

The three AFPs used in the Advanced Document Retrieval System are high-performance digital processors and work together in the AFP Array Computing System. The basic hardware elements of an AFP Array Computing System are an array of AFPs, a high-performance random access memory, and a host computer. High data processing rates are achieved by having parallel architecture, multiple processors, and instruction parallelism internal to the processor. Applications execute entirely within the AFP Array Computing System.

The software structure necessary to control the AFP Array Computing System is based on the allocation of work in the system. The primary work is to perform compute intensive data processing. In the AFP Array Computing System, the processor array performs this function. In order to manage the high-speed processing, low-speed scheduling and support functions must be provided. The host computer supplies these low-speed control and support services, such as loaders, debug tools, and interactive and batch interfaces.

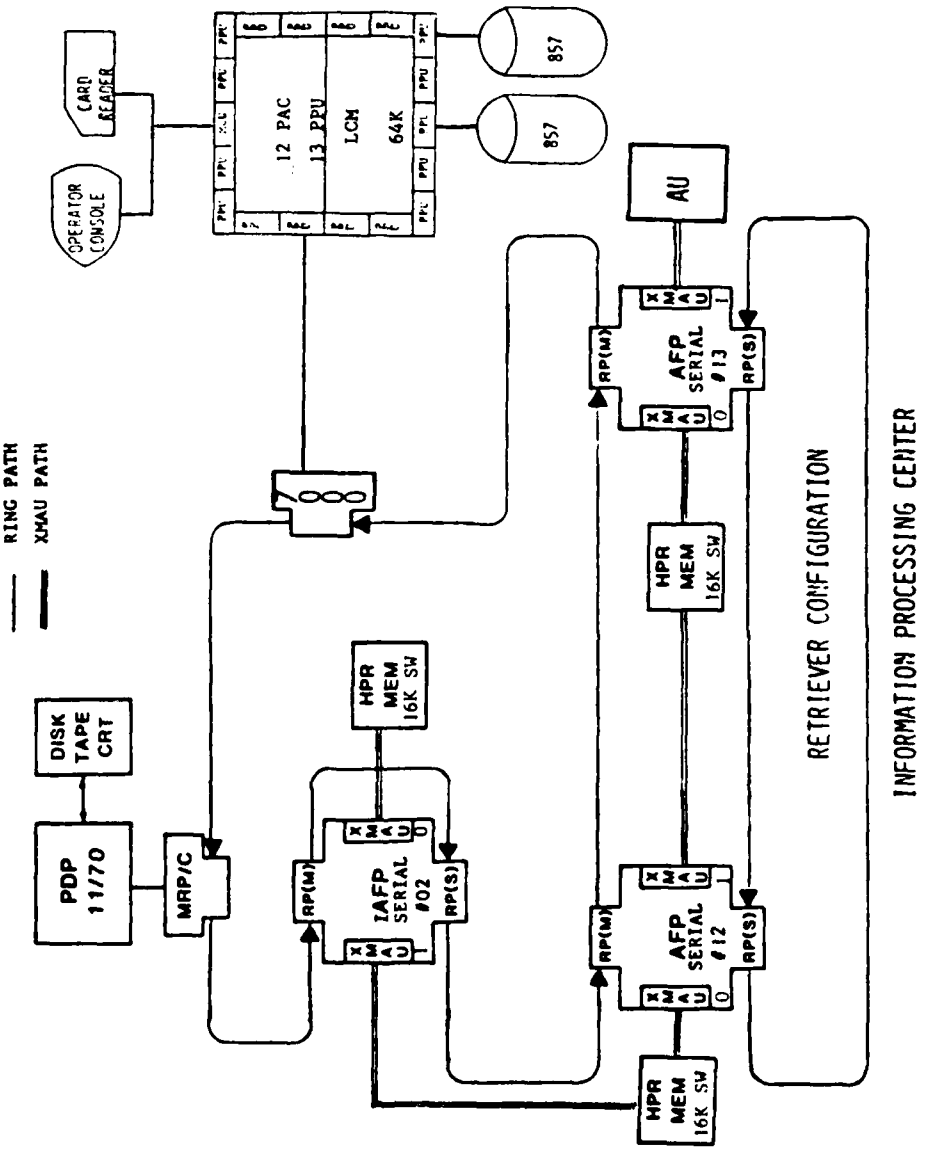


Figure 3-1. System Configuration

3.1.1 Hardware Structure

The connection between the host and the AFP array is a high-speed ring channel with a ring port to the host. The connection between the AFP array and the high performance RAM (HPR) memory is an extremely high-speed direct memory, access-type transfer.

An AFP consists of a collection of relatively autonomous functional units interconnected by a crossbar switch. An instruction memory controls these functional units as well as the configuration of the crossbar switch each machine cycle. The functional units in a processor operate in a synchronous manner, with most units capable of producing results every machine cycle.

Interprocessor communications in an AFP array are performed with a ring communications system in which packets of information containing both data and control are transmitted between processors. Packets are passed from processor to processor over this parallel channel until removed by the destination ring port. Ring bandwidth is a function of the number of rings and the number of processors on each ring.

The AFP consists of 15 functional units, connected via a crossbar switch. Figure 3-2 shows the AFP crossbar configuration. The crossbar switch can route as many as 16 16-bit input quantities to as many as 18 destinations, on any clock cycle. The AFP contains eight different functional unit types:

- Data Memory (4),
- Integer Add (2),
- Multiplier (1),
- Shift/Boolean (2),
- File (1),
- Ring Port (2),
- External Memory Access (2),
- Control (1).

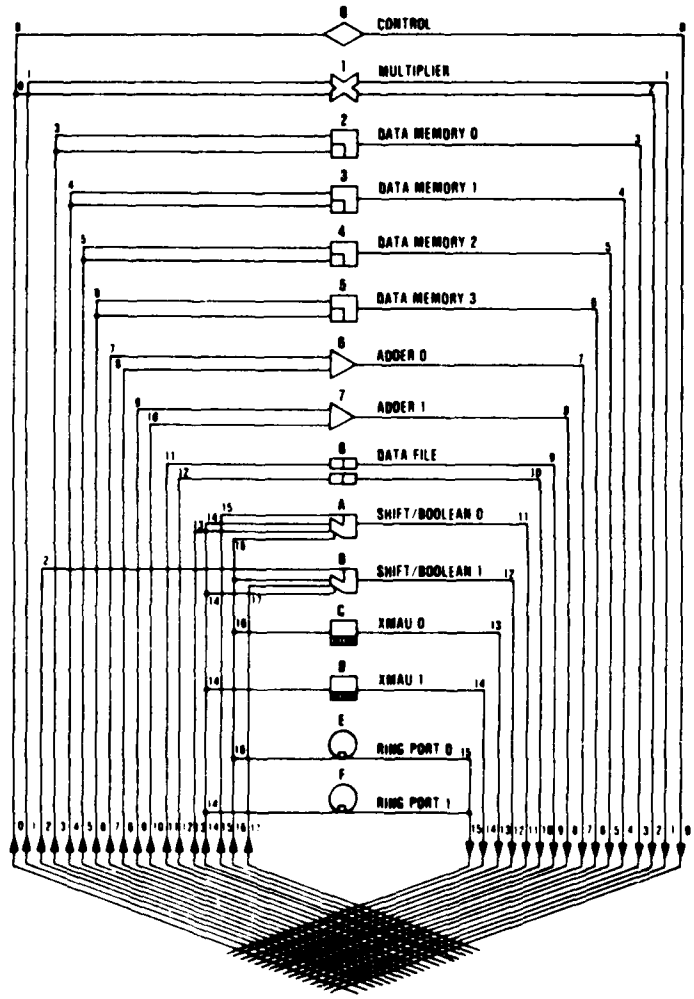


Figure 3-2. AFP Crossbar Configuration

The primary capabilities of the hardware are summarized in Table 3-1.

The processor operates in a synchronous manner, with a cycle time of 20 nanoseconds. Each unit contains input registers controlled from the micromemory instruction word. All units are segmented and capable of initiating a new operation every cycle. Most units perform their operations and deliver results to the input registers of other functional units in two machine cycles. Multiplication operations require one extra cycle. The control unit and the input/output units take one cycle for immediate operations. Each unit does comparisons of the results of the operation performed. These comparisons are available at all times for use in branching or decision making.

3.1.2 Software Structure

The software structure for the AFP system consists of three subsystems. The PDP-11/70 control subsystem software performs AFP array initialization, control and scheduling, and termination. The AFP array subsystem software provides executive and control services to the application code executing in the AFP. The general-purpose computer subsystem software provides the software development tools.

PDP-11 Control Subsystem Software

The scheduling and control of application functions running in the array are managed by the Multiprocessor Array Executive (MAX) operating system. MAX consists of a number of tasks executing in close cooperation with, and under control of, the RSX-11M operating system. These tasks provide interactive and batch interfacing services for the application function, the I/O services to the AFP array, and the executive features for overall control of the system. The controlling element resides in the PDP 11/70.

TABLE 3-1. AFP FUNCTIONAL UNIT CAPABILITIES

INTEGER ADD	ADD/SUBTRACT, 16 OR DUAL 8-BIT, 1's OR 2's COMP, 32-BIT NETWORK
MULTIPLY	16 x 16, DUAL 8 x 8, INTEGER BYTE PRODUCTS, 2's COMP, POPULATION COUNT, SIGNIFICANCE COUNT, BIT REVERSAL
SHIFT/BOOLEAN	32-BIT RIGHT OR CIRCULAR SHIFT, 16 BOOLEAN OPERATIONS
FILE	2 SETS OF 8-WORD x 16-BIT REGISTERS, 2 16-BIT READS AND 2 16-BIT WRITES PER CYCLE
DATA MEMORY	1024 16-BIT WORDS, 16 16-BIT REGISTERS DIRECT AND INDIRECT ADDRESSING, AUTOMATIC INDEX INCREMENT/DECREMENT
CROSSBAR PORT	16-BIT OUTPUT AND 16-BIT INPUT DATA CONNECTIONS TO THE CROSSBAR
RING PORT	16-BIT DATA I/O WITH RING AND PROCESSOR, 16-WORD INPUT AND 16-WORD OUTPUT BUFFERS FORCED TRANSFER TO ALL PROCESSOR MEMORIES
XMAU	EXTERNAL MEMORY ACCESS 128-BIT MEMORY I/O AND 16-BIT PROCESSOR I/O, MAX ADDRESS 3 BYTES

The basic unit of work for the AFP Array Computing System is the application program, which is the combination of PDP-11/70 (control program) and AFP software that is needed to do an application task. The executable software for applications is stored under the RSX-11M operating system. The structure permits existing modules to be easily changed or new ones to be added. The application control program is written as a normal RSX-11M task that uses the special MAX interfaces. It is written in FORTRAN or MACRO 11 (PDP-11/70 assembly code), and is constructed from relocatable routines to interface to the MAX services. The AFP array portion is written in MICA language and uses the AFP executive and resident services. The control program is responsible for managing the loading of the AFPs and for high level control of the application. The MAX system is designed so that an application may execute as either a batch job or an interactive job. Procedure files may be used to execute sequences of commands by calling a file rather than entering the commands one at a time.

A DEBUG software package is available to provide a means of monitoring the AFP Array Computing System while executing user applications or engineering diagnostics, and to provide tools for tne user or engineer to manipulate the system as an aid to program debugging or fault finding. A comprehensive set of commands is available to enable the user to query an operational system. Different displays are available to view the system state and the state of any individual processor or memory.

AFP Array Subsystem Software

The AFP array subsystem consists of one interface AFP (IAFP), and several AFPs (typically 2 to 31). The AFP executive software resides only in the IAFP and communicates to the other AFPs through the AFP resident software. It accepts requests from the AFPs and MAX. High-level requests from the MAX system task or application task are passed to the

AFP executive by the ring driver. The AFP executive expands these requests into the necessary detailed level and sends ring packets over the ring to the AFP resident programs in the AFPs. In order to access basic ring interface features of the AFP, the AFP executive can permit a transfer of unmodified ring packets to the AFPs.

The AFP resident performs various system functions required by the AFP application program and receives requests from the AFP executive through the system ring. The AFP resident resides in each AFP (except the IAFP) along with the user AFP application program.

General-Purpose Computer Subsystem Software

The software development tools provided by the general-purpose computer software are the AFP cross assembler, MICA, and the AFP instruction level simulator, ECHOS. The MICA cross assembler and the ECHOS instruction level simulator allow all programming to be done off-line. Any Control Data CYBER 700/800 series computer hosts these software tools.

AFP programs are written in the MICA language on the CDC CYBER computer. The edited files are then assembled by MICA. MICA checks for all illegal syntax and illegal hardware usages. Functional unit and crossbar usage conflicts are identified by MICA. A binary file is produced in the format required to be loaded in the AFP.

The AFP simulator executes on a CDC CYBER computer and provides a detailed simulation of all AFP functions on a clock cycle basis. ECHOS provides a set of commands for controlling the loading and simulated execution of AFP microcode programs output by the MICA cross assembler. ECHOS may be used interactively or in batch mode. In batch mode, the commands controlling the simulation and its output are read from a file and all output is written to a file. In interactive mode, the commands

are typed in directly by the user and the output is returned to the user at the terminal. This gives the user flexibility in the control of the simulation process. In addition, the user may direct all or part of the output, with a copy of the input, to a file for later off-line examination. ECHOS takes the binary microcode program directly from a file created by the MICA cross assembler.

3.2 HOST COMPUTER

The host computer of the Advanced Document Retrieval System is a DEC PDP-11/70. It is responsible for hosting the application and system software. The AFP system software consists of an Executive, MAX, a debug package, and diagnostics. The host also supports all the standard input and output functions of the system. The application software is written to be executed in the various processors and stored on the host computer.

3.3 ASSOCIATIVE MEMORY UNIT

The Associative Memory Unit consists of 256 associative processing cells and a microprogrammable controller. The exploratory development model developed during the previous contract was expanded and an interface to the AFP was designed and built. The following paragraphs provide an overview of the Associative Unit and a more complete description is found in Appendix C.

The Associative Memory Unit contains 256 associative processing cells designated cell 0 through cell 255. A lowered numbered cell is considered to be above a higher numbered cell, and the lowest numbered cell of a collection of cells is considered to be the first cell in that collection.

All cells in the Associative Memory Unit are common to four buses which carry data and control to the cells and data from any one of them. In addition to the common bus connections, each cell has a set of unique connections for propagating data between adjacent cells, and for future connection of external devices. Each cell also has connections to the response network.

Each cell is in one of two states: marked or not marked. The response network deals only with the marked states of each cell. In addition to the marked state, several other status conditions are defined within each cell. Various cell operations may be conditioned on the true or false value of these conditions. The elements of a cell are illustrated in Figure 3-3 and include a 256-words by 4-bit random access memory (RAM), a 32-function, 4-bit arithmetic logic unit (ALU), cell control logic, and various registers and internal buses. The cell operations are controlled by the Cell Control Bus (CCB).

The Associative Processing Cells are controlled by a microprogrammed controller. The controller is connected to the External Memory Addressing Unit (XMAU) of the AFP. The interface is designed to make the Associative Memory Unit appear to be a bank of High Performance Memory (HPR). By initiating a read or write to the Associative Memory Unit, the AFP can load or read back the microprogram memory, examine the registers, select the operating mode, start a microprogram, or transfer data to and from the Associative Memory Unit.

The controller consists of a 256-word by 48-bit microprogram memory, several registers, and associated interface and control logic. A block diagram showing the functional organization of the controller appears in Figure 3-4.

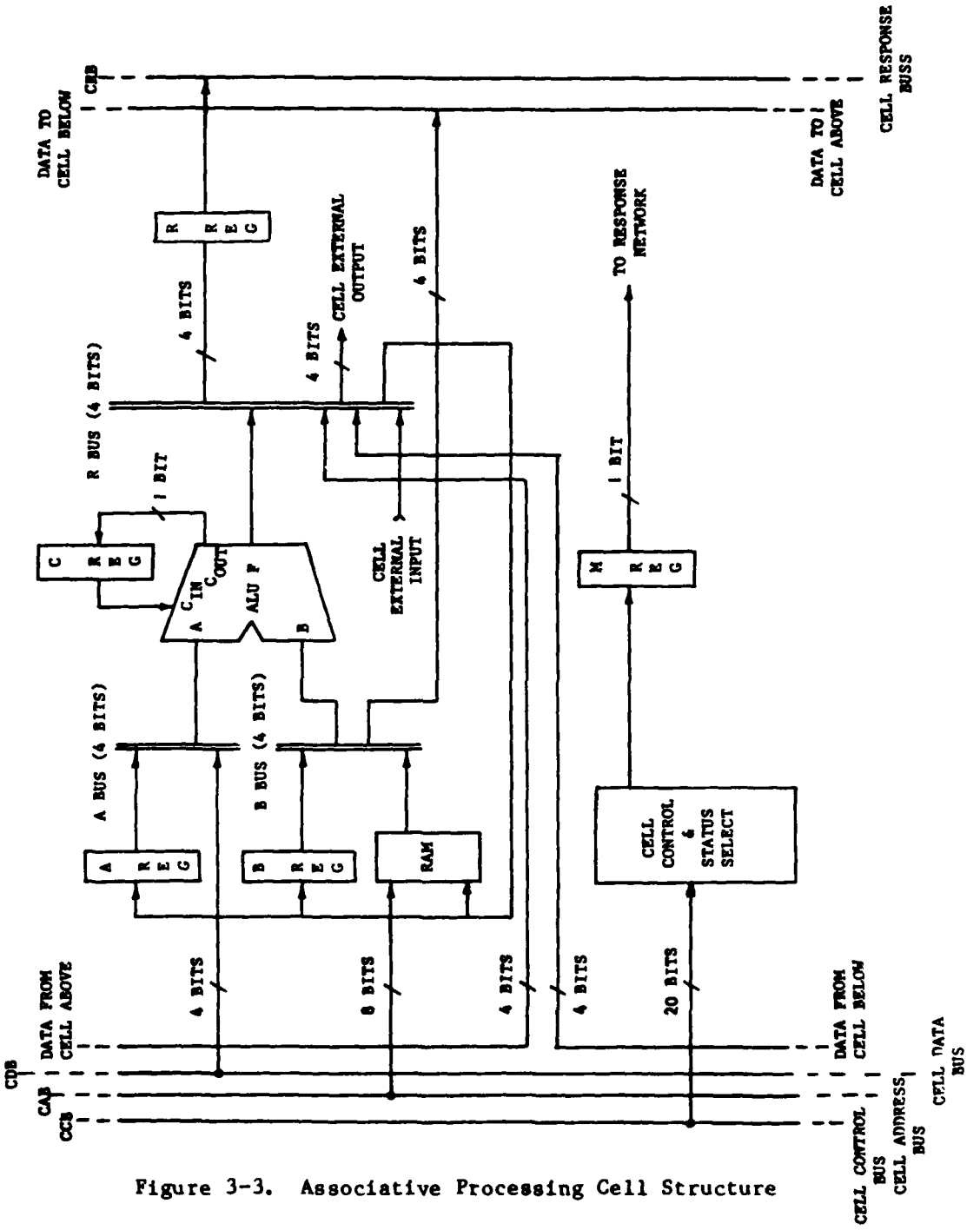


Figure 3-3. Associative Processing Cell Structure

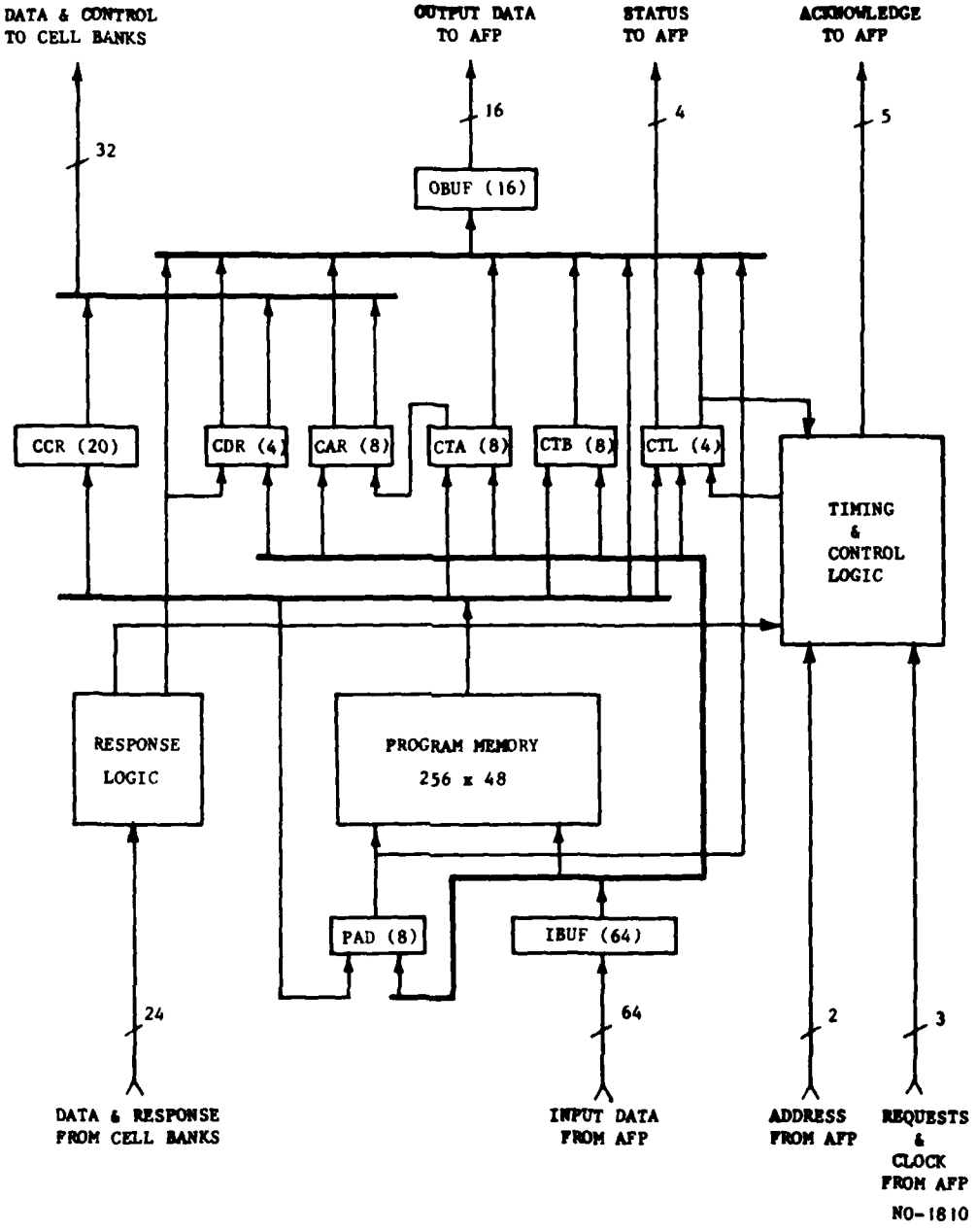


Figure 3-4. Associative Unit Controller Block Diagram

3.4 DISK STORAGE SUBSYSTEM

The Disk Storage Subsystem (DSS) contains 13 CDC 7600 Peripheral Processors, 320K 12-bit memory words, two CDC 857 disk drives, a CDC 405 card reader, an operator console, and a CDC 7000 channel to an AFP ring port interface. Figure 3-5 shows the DSS configuration and the interface to the complete Advanced Document Retrieval System.

The Disk Storage Subsystem is attached to the AFPs via the ring channel. This also allows the host computer to communicate with, and transfer data to/from the DSS.

Four of the thirteen peripheral processing units (PPU) are used. Four unique tasks are performed and each has a dedicated PPU. The four tasks are: the system monitor, the interface and command PPU, and two disk drivers. Appendix D details the features of the interface and command PPU and disk driver PPU.

The PPUs are separate and independent computers. Each PPU has a computation section that performs binary computation in fixed-point arithmetic. A PPU provides storage for 4096 12-bit words. The PPU instruction set, combined with the high-speed memory and channel flexibility, enables a PPU to drive many types of peripheral without the necessity of an intermediate controller. There are eight input data paths and eight output data paths connecting the PPU to other devices. The PPU input/output facility provides a flexible arrangement for high-speed communication with a variety of I/O devices. PPU to PPU communications is performed through dedicated addresses in main memory.

Main memory consists of 320K of 12-bit words or 64k of 60-bit words. A read from or write to memory is eight 60-bit words in parallel. A PPU disassembles (reads) or assembles (writes) these eight 60-bit words in 12 bit increments.

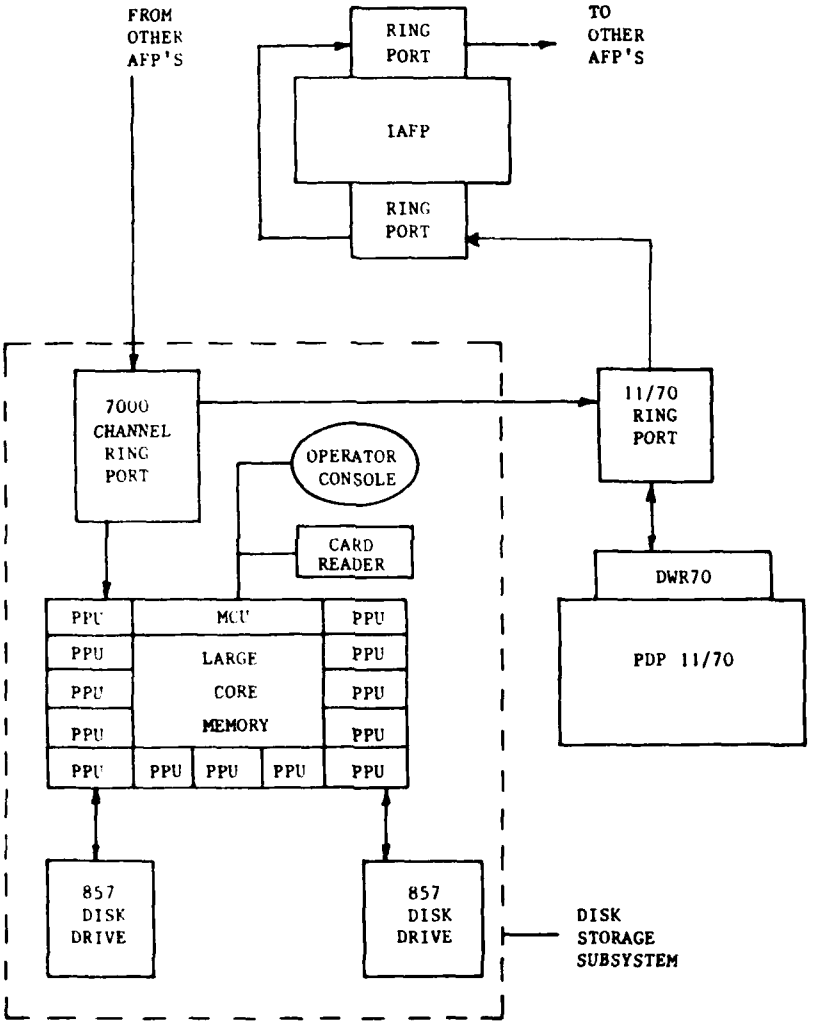


Figure 3-5. Disk Storage Subsystem

4.0 DEMONSTRATION SOFTWARE

The software developed for the Advanced Document Retrieval System is partitioned among the several programmable hardware components of the system to provide a highly flexible interactive document retrieval system. The software system was organized to readily facilitate modification and growth, and was partitioned to specific hardware units to take advantage of their specialized processing capabilities. The software developed in this program includes:

- o Host software
  - Written primarily in Fortran and operates on the host computer, a DEC PDP-11/70. This software processes the interactive user commands, generates the user display, and controls the network of AFPs and the AU.

- o AFP software
  - Written in MICA assembly language for the network of three AFPs. In general this processes the high-speed operations. Appendix B, "AFP Description" describes the programming characteristics of the AFP.

- o AU microcommands
  - This microcode is absolute binary code for controlling the associative unit. The microcode formats are described in Appendic C, "Associative Unit" and the annotated microcode is listed in Appendix E "Associative Unit Microcode for Demonstration". For this demonstration the AU microcode consists of a total of 59 instructions in 5 routines.

- o Disk controlware
  - This is assembly code operating in the peripheral processor units. This code provides system monitor ring channel adapter control, and disk control functions. This controlware is described in detail in Appendix D, System Software Description.

Much of the software and microcode identified above is transparent to the user and will not be described in this section. Rather, this section emphasizes the organization and general implementation approach for the software supporting system user functions. Further details on the user-visible software are given in Appendix A, "Demonstration Document Retrieval System Users Manual".

4.1 SOFTWARE ORGANIZATION

The Advanced Document Retrieval System shows the feasibility of using an Associative Unit and an AFP system to perform data compression of a set of many text files for the purpose of forming a document retrieval data base. It includes software to create the data base, to search the data base for user specified phrases, and to display the documents containing the phrases. The software package used to support this demonstration is named RETRIEVER.

The hardware configuration includes a PDP 11/70 as the host computer to an AFP array. An Associative Memory Unit is attached to one of the AFP's for the data compression function.

The software development was executed in two phases. The first produced a PDP 11/70 only software system, where all of the functions of data compression, document search, and text retrieval were performed by Fortran code. This demonstrated the feasibility of the data compression algorithm.

The second phase resulted in AFP software and Associative Unit microcode for data compression as well as AFP sorting of the dictionary and occurrence lists. The software structure description in this section is for the AFP and Fortran implementation. Additional details on the software are included in Appendix A, "Demonstration Document Retrieval System Users Manual" and Appendix D, "System Software Description".

The software system is organized as an interactive command language processor. This command language, specifically developed for the demonstration system is called Retriever Command Language (RCL). Figure 4.-1 shows the top level organization. The software provides for initialization, termination and menu command processing. This menu command processing provides the basis for user interaction for processing the four major applications handled by RCL:

1. Creation and storage of data bases of documents.
2. Expansion (merge and purge) of data bases.
3. Retrieval of documents that satisfy specified criteria.
4. Scanning of retrieval document texts.

4.1.1 Terms and Abbreviations

1 MAX -- Multiprocessor Array Executive
MAX is a PDP 11/70 and AFP software system for controlling the AFP array, for providing software services to access and utilize the array, and for providing development services for applications (debug tools).

2 HPR -- High Performance Random Access Memory
HPR is a dual-ported memory bank with 16K x 64-bit superwords (swords). The Advanced Document Retrieval System configuration has 3 banks of HPR. One bank is used by the interface AFP. A second bank is shared by the interface AFP and applications AFP-1. A third bank is shared by applications AFP-1 and applications AFP-2.

3 AU -- Associative Unit
The AU is an "intelligent" micro-programmable memory device that performs simultaneous operations on a data dependent subset (or all) of its storage locations.

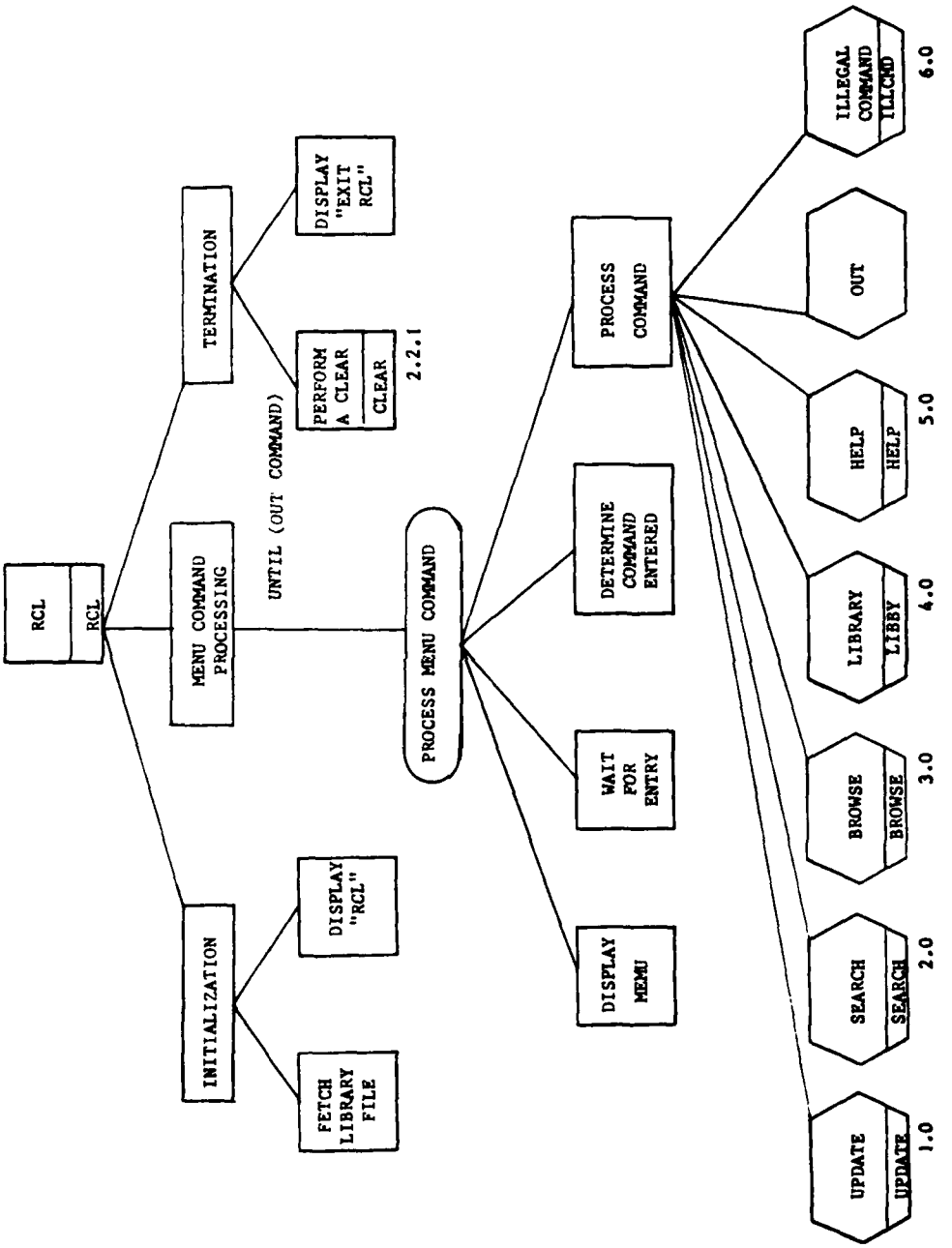


Figure 4-1. Advanced Document Retrieval System Software Organization

4.2 System/Subsystem Description

The Advanced Document Retrieval System includes software to create a document retrieval data base, to search the data base for occurrences of user defined phrases, and to display the text of the documents containing the phrases.

The RETRIEVER software was built using the RSX-11M operating system on the PDP 11/70 and the MAX software system for the AFP array.

The central functions of the RETRIEVER document search system are:

- o INPUT
  - PDP 11/70 and AFP code create a data base consisting of a dictionary and an occurrence list.

- o SEARCH
  - The user specifies a word or phrase and the dictionary is searched for all occurrences of the phrase. A list of the test documents that contain the phrase is produced.

- o BROWSE
  - The text for the documents that contain the specified phrase were displayed to the user, one document at a time.

The software also supports auxiliary functions which include:

- o MERGE
  - The occurrence list and dictionary of two data bases can be merged together to form one larger data base.

- o RENAME
  - The title of a data base can be changed.

- o REMOVE
  - A data base can be eliminated from the library.

- o LIBRARY
  - A list of data bases, the number of documents, and the amount of disk resources consumed is displayed.

4.2.1 Equipment Environment

The RETRIEVER Software System uses 3 computer types and a variety of peripherals. A list of computers and the important peripherals are:

1. PDP 11/70
   a. Terminals -- for human interaction
   b. Disks -- for text file storage
   c. Tape Transport -- for data input

2. AFP -(Interface AFP)
   a. HPR -- 1 bank for the use of the interface AFP
   b. HPR -- 1 bank shared by the interface AFP and AFP-1. This bank provides the data input capability.

3. AFP-1 This AFP performs sort, merge, and dictionary build
   a. HPR -- 1 bank shared by AFP-1 and AFP-2 for sharing data

4. AFP-2 This AFF hosts the Associative Unit

5. Associative Unit Performs data compression and stopword removal

6. 13-Pack A 13-PPU subsystem connected to AFP RING
   a. CYBER 7000 ring port -- connected AFP ring to 13-pack channel
   b. Disks -- retain occurrence lists and dictionary

4.2.2 Support Software Environment

The MAX (Multiprocessor Array Executive) system is a PDP 11/70 and AFP software system for controlling the AFP array, for providing software services to access and utilize the array, and for providing development services for applications (debug tools). It includes resident microcode in each applications AFP, an interface AFP between the PDP 11/70 and the ring, an RSX-11M I/O driver, interactive control software, scheduling software, and a subroutine library for use of the applications programmer. RSX-11M is the operating system for the PDP 11/70.

4.2.3 Interfaces

The input interface for RETRIEVER is through magnetic tape containing 5000 byte fixed length records. Each physical record contains one or more logical records.

The RETRIEVER software resides in the PDP-11/70, the AU, and the AFPs. The MAX system interfaces the three types of devices through the RSX-11M I/O driver for the ring interface, the interface AFP software, the AFP resident software in the application AFPs, and the AU resident software.

The interface of the PDP 11/70 Fortran to the applications AFP microcode is through the MAX subroutine library.

The interface AFP communicates with AFP-1 and AFP-2 through the ring. AFP-1 and AFP-2 communicate via the ring and the shared HPR. AFP-2 communicates with the AU through an XMAU port.

The AFPs communicate with the 13-pack disks through a CYBER 7000 channel ring port (hardware) and PPUs on the 13-pack system. One PPU controls the ring port, one PPU controls each of the 13-pack disks, and one PPU controls System Maintenance Monitor (SMM) for the 13-pack display.

4.3 DESIGN DETAILS

4.3.1 Data Base

The following data files are used by RETRIEVER

1. Magnetic Tape
   The input for creating the data base is a magnetic tape containing 5000 byte physical records. Each physical record contains one or more variable length logical records. Each logical record contains 3 bytes that identifies the record type and the length of the record. The records other than the text data are called the formatted fields. The record types include the following.

   000 -- I.D.
   An 8 character document identification code

   006 -- Title
   Title of the document

   037 -- Date
   Date of publication

   101 -- Author
   Names of authors

   103 -- Country
   Country of publication

107 -- Institution
Names of the institutions supporting the publication

143 -- Text
Text of the document (Each document had one or more text records.)

2. Text File
The text file is retained on the PDP 11/70 disks. It is a separate file from the formatted field data.

3. Formatted Field File
The formatted field file contains the non-text records of a document. It also contains the index into the text file for the start of the document. If the amount of formatted field data is small, the information and pointer for up to 4 documents could be in one 512-byte PDP 11/70 disk sector.

4. Segment Table
The segment table is the last block of every dictionary file on the 13-pack disks. It provides a quick lookup for the dictionary. The table has one entry for each dictionary block. It is the last word in the dictionary and the address of the dictionary block. When a search of a data base is made, the segment table is read first and retained in memory, then the correct dictionary block is used for the word being searched.

5. Dictionary
The dictionary resides on the 13-pack disks. The dictionary begins with the highest disk address and uses successively decreasing disk addresses. Each dictionary entry consists of 6 16-bit words and includes the following:

   a. Compressed code for the word in 9 8-bit bytes. If the word compresses to 9 or fewer bytes, all bytes are retained; however, if the word compresses to 10 or more bytes, 8 bytes are retained and the 9th byte is a marker indicating the word compresses to 10 or more bytes.

b. Block number (on 13-pack disks) of the beginning of the occurrence list for the word

c. Sub-block number of the beginning of the occurrence list

d. Entry number within sub-block of the beginning of the occurrence list

e. Number of occurrences

6. Occurrence List
The occurrence list resides on the 13-pack disks. Each occurrence list entry consists of 2 16-bit words. The occurrence list includes the following:

a. Document number. This is a pointer to the formatted field file on the PDP-11/70 disk. It is not just a sector in the formatted field file for the document (upper 14 bits) and the index of the section of the sector (0-3 in lowest 2 bits) for the document.

2. Sentence number. Sentences are marked by periods, '.', in the text. The sentence number indicates the order of sentences within each document.

3. Word number. The word number within the sentence.

7. Result List
The result list file is produced on the PDP 11/70 disks as a result of the search operation. When the user specifies a new phrase to search for, a new entry is created in the search list file. Each entry contains the numbers of the documents that satisfy the search request.

The pointers of the 13-pack entries are based upon the 13-pack memory and disk sector sizes. Each large core memory (LCM) word of the 13-pack memory is 480 bits, or 30 16-bit words. Each disk sector is 51

LCM words. A track contains 4 sectors or 204 LCM words. In the dictionary entry, the block number is the track address. The sub-block number is the LCM word within the track (0-203), and the entry number is the 16-bit pair within the LCM word (0-29).

4.3.2 Program Descriptions

Data Base Creation

The basic flow of the data base creation through the RETRIEVER software is as follows:

1. The PDP-11/70 software
   a. Read the documents from the magnetic tape,
   b. Copy the textual data to a PDP-11/70 disk file,
   c. Split the formatted field data (author, institution, date) to a PDP-11/70 disk file,
   d. Filter the text for illegal characters,
   e. Transmit the input data to the AFP-1 HPR while appending a document for later retrieval of the text data.
2. AFP-1 transfers the data from its HPR (where the PDP-11/70 wrote the data) to AFP-2 for compression by the AU.
3. AFP-2 calculates sentence number and word number for each word.
4. AFP-2 passes the data to the AU for compression.

5. AU searches (simultaneously) its memory for a match to initial sub-strings of the four characters and returns the compression character for the longest n-gram.

6. AFP-2 sends complete compressed words to the AU for stopword testing.

7. AU searches (simultaneously) its memory for a match to the complete compressed word. If a match is found, the AU marks the word as a stopword and it is not put in the dictionary.

8. AFP-2 passes the compressed word back to AFP-1. Each word includes a document number, sentence number, and word number. These numbers are necessary for searching. AFP-1 then builds up a buffer in HPR of the compressed words. After the buffer is filled, the buffer is sorted before it is written to disk. AFP-1 and AFP-2 continue to input, compress, sort, and output buffers to disk until all input is completed.

9. AFP-1 merges occur after all data is input. It is a 1, 2, or 3-pass merge for up to 8, 64, or 512 blocks, respectively. AFP-2 does the 8-way compare for the merge, under control of AFP-1.

10. AFP-1 builds the occurrence list and the dictionary after the merge phase is finished. The output of the merge phase is a sorted list, where every word of every input document is listed in compressed form with its associated tag to recover the original text. Since this is too bulky for permanent storage, the AFP-1 builds a dictionary where the pointer to the occurrence list, and the number of occurrences of the word are maintained. The occurrence lists contain no compressed data, but only the tags in the order of the sorted data. Since duplicate words are adjacent in the sort list, it is necessary only to know the beginning of the set of adjacent tags, and the number of tags for a given word.

11. The demonstration system has 2 disks, so the occurrence list begins at the lowest address of drive 0 and uses successively increasing addresses. The dictionary begins at the highest address of drive 1 and uses successively decreasing disk addresses. This reduces head movement during the search phase.

12. AFP-1 also contains the code to read and write the 13-pack disks as part of the sort and the merge phases.

The PDP-11/70 Fortran creates a file of the original text data on the PDP-11/70 disks. It does not contain the formatted fields and the characters are padded so a word does not cross a sector boundary.

The PDP-11/70 Fortran also creates a formatted field file. This file contains the fields indicating the author, institution, and date. They do not become part of the compressed text. More importantly, the file contains a pointer to the text data within the text file. The document number that is passed to the AFP is the index into this formatted field file.

After the data base creation is completed, the PDP-11/70 code adds the data base name to the library. This is an PDP-11/70 disk file that listed the library name, the number of documents, the addresses of the occurrence list and the dictionary on the 13-pack disks, and the number of sectors used by each.

Data Base Merge

The merge of two data bases is done by PDP-11/70 code. The following steps show the basic flow.

1. Copies the formatted field file and the text file to new files that are to be associated with the merged data base.

2. Appends the formatted field file and the text file of the second data base to those of the first data base.

3. Reads beginnings of the dictionary lists, and the starts of the occurrences lists from the 13-pack via AFP-2.

4. Merges the dictionaries and occurrence lists. The occurrence list entries for the second data base are modified to point to the latter part of the text file, where the second data base text file is appended to the text of the first data base.

5. Writes the dictionary and occurrence lists back to the 13-pack disks via AFP-2.

Data Base Rename

The PDP-11/70 code renames a data base by changing the entry in the library file. It also renames the text and formatted field files on the 11/70 disk, since these file names are based upon the data base's name.

Data Base Removal

The PDP-11/70 code removes a data base by deleting the 11/70 text and formatted field files, and by removing the data base entry in the library file. The data on the 13-pack disks is not shuffled to create larger blocks of free space; nor is the space of the now-useless occurrence list and dictionary made available for later use.

Library Display

The PDP-11/70 code formats and labels the data of the library file. The library display includes the data base name, the number of documents, the starting sectors of the occurrence list and the dictionary, and the number of 13-pack sectors for each.

Search

The search process is implemented in PDP-11/70 Fortran and uses AFP-2 to read the 13-pack disks. The processing begins by parsing the search phrase into a series of strings with conjunctives. The possible conjunctives include the following.

1. AND
   The two words appear in the same document.

2. OR
   Inclusive or. One word, or the other, or both are in the document.

3. ADJ
   The two words are adjacent in the document.

4. SEN
   The two words are in the same sentence in the document.

5. NOT
   The sense of a logical expression in the search phrase is negated.

6. IMPLIED ADJACENCY
   The two words are specified with no explicit conjunctive, and adjacency is assumed.

7. (-)
   Parenthesis may be used to group operations.

The operators work on phrases as well as words, if the phrases are enclosed in parenthesis.

The search expressions are evaluated from left to right. Expressions inside parenthesis are evaluated first. ADJ has a higher priority than SEN, which has a higher priority than OR, NOT, and AND.

The search command requires the user to specify the data base name to be searched. It then initializes the results list and assigns result numbers to successful matches. The satisfying document list is retained with the result number as the index. Subsequent browses require the result number to be specified.

Multiple searches could be done before browsing. Each phrase is assigned a successive result number.

The search processing begins with parsing the search phrase into words with conjunctives. The processing begins with the most deeply nested phrase (most parenthesis). If more than one phrase has equal nesting, the left-most phase is processed first. A stack is used to retain the results of operations.

The segment table for the data base is read from the 13-pack disks via AFP-2, at the beginning of the search. Then, for each word, the segment table entries identify the dictionary track to read for the word being sought. Finally, the occurrence list is read. The occurrence lists of two adjacent entries in the search phrase are processed according to the conjunctive of the entries, and an output occurrence list is formed. If further processing is to be done because of nesting or further entries in the search phrase, the result is placed on a stack. The resultant lists are processed from the stack until one list is finally formed. The final list is placed in the resultant file for the browse activity.

The search command also displays a count of the occurrences and the number of documents that satisfy the search activity for each phrase.

Browse

The browse command is implemented in PDP 11/70 Fortran and requires only the resultant list file, the formatted field file, and the text file on the PDP 11/70 disks.

The browse function displays the documents that satisfied previous searches. Each successful search has an index number. When browsing, the user specifies the index number. For each document that satisfies a search, the browse command allows the user to see the formatted fields, or the fields and the text.

The browse function begins by opening the resultant list file. It then uses the document number to index into the formatted field file. This file provides the field information as well as a pointer to the text in the text file. The formatted fields are then displayed and the user is asked if he wants to see the text. If selected, the text is then scrolled to the screen. The fields and the text are displayed for each document in the resultant list until the end of the list or until the user cancels the browse.

5.0 DEMONSTRATION

The purpose of the Advanced Document Retrieval System Demonstration was to show the capabilities of the multiple AFP system and the Associative Unit in a document retrieval system in a simulated, operational environment.

5.1 DEMONSTRATION PLAN

There were two phases to each pass of the demonstration. The first phase executed a large batch file simulating the interaction of a user over a long period of time. The second phase was the hands-on evaluation of the system.

This batch file demonstrates, in a very timely fashion, the features of the documentation retrieval system. The MAX operating system of the AFPs has been designed to support both batch and interactive jobs, allowing the job type to be transparent to the features of the operating system software. This batch file was created for rigorous testing of the operating system software in a consistent manner. Testing was not sidetracked by unpredicted results caused by typing errors or slowed down by the manual entry of the same tests during software testing. This batch file worked so well during test, it was added to the demonstration to show the completeness and features of the system.

5.2 UPDATE DEMONSTRATION

Four functions apply to the creation or modification of the document data bases. They are:

1. Creation of a data base,
2. Merging of 2 data bases into a third
3. Renaming of a data base,
4. Removal of a data base.

The update demonstration can:

- o Create 3 data bases
- o Dump occurrence list sectors
- o Dump dictionary sectors
- o Dump segment table
- o Merge 2 of the data bases into a fourth data base
- o Rename the second data base
- o Delete the third data base
- o Create a fifth data base

O - the first 3 data bases are created, 4 occurrence list sectors, 4 dictionary sectors, and the segment table are dumped to a PDP-11/70 file and to the line printer. The dictionary entries illustrate the text compression algorithm. The segment table is examined for correct references. The occurrence list format is illustrated. The first 4 data bases use a short 11/70 file as input or a limited number of tape records. The rename, delete, and merge functions use the library display for verification of the actions.

The fifth data base is created from tape and has over 3000 documents.

5.3 SEARCH DEMONSTRATION

The search overlay searches the specified document dictionary for the specified phrase. A search phrase may be one or more words joined by the operators.

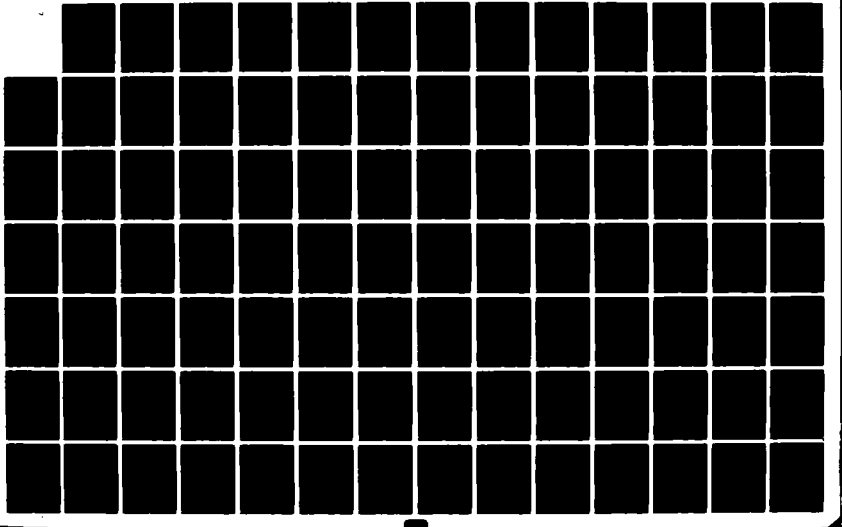
ADJ	adjacency (usually implied) The two words follow each other in the same sentence.
SEN	sentence The two words are in the same sentence.
AND	and The two words are in the same document.
OR	or One or the other or both words are in the same document.
NOT	not Only one word is in the document.

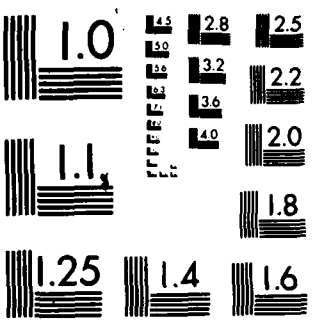
The operators may apply to phrases as well as words if the phrases are contained within parentheses. For example, the search line '(NICKEL OR CADMIUM) ADJ STEEL' searches for 'NICKEL STEEL' or 'CADMIUM STEEL'. Words that do not have an explicit operator between them are implied to be adjacent, for example, 'TURBINE BLADES' which means TURBINE ADJ BLADE. However, an explicit operator must precede and follow parentheses groups.

Similar words that differ in suffix only may be used interchangeably if the dollar sign, '$', is specified. For example,

AD-A139 385 ADVANCED DOCUMENT RETRIEVAL SYSTEM(U) CONTROL DATA CORP 2/3
MINNEAPOLIS MN W CYRE ET AL. JUL 83 RADC-TR-83-168
F30602-79-C-0231

UNCLASSIFIED F/G 9/2 NL





DOCUMENT$ may be interpreted as

DOCUMENT
DOCUMENTS
DOCUMENTATION
DOCUMENTARY
etc.

The '$' may be used at any position in the word. The word, 'C$', is all words in the dictionary that start with a C.

The demonstration of the SEARCH overlay will begin with single word searches and proceed to complex phrases in four stages.

The first stage demonstrates that all words of a document, except for the stop words, are properly catalogued. A document is selected at random and a search is conducted for every word in the document. Visual inspection is used to verify that the occurrence list contains the chosen document.

The second stage contains 2-word phrases where the 2 words are joined by implied adjacency, explicit adjacency, SEN, AND, OR, or NOT. Four phrases are chosen at random from the data text.

The third stage contains 2-word phrases joined to a third word by an operator. The patterns are as follows.

```
  X     Y       Z
( X     Y )     Z
( X     Y ) ADJ Z
( X     Y ) SEN Z
( X     Y ) AND Z
( X     Y ) OR  Z
( X     Y ) NOT Z
```

```
  X ADJ Y   ADJ Z
( X ADJ Y )     Z
( X ADJ Y ) ADJ Z
( X ADJ Y ) SEN Z
( X ADJ Y ) AND Z
( X ADJ Y ) OR  Z
( X ADJ Y ) NOT Z


  X SEN Y   SEN Z
( X SEN Y )     Z
( X SEN Y ) ADJ Z
( X SEN Y ) SEN Z
( X SEN Y ) AND Z
( X SEN Y ) OR  Z
( X SEN Y ) NOT Z


  X AND Y   AND Z
( X AND Y )     Z
( X AND Y ) ADJ Z
( X AND Y ) SEN Z
( X AND Y ) AND Z
( X AND Y ) OR  Z
( X AND Y ) NOT Z


  X OR  Y   OR  Z
( X OR  Y )     Z
( X OR  Y ) ADJ Z
( X OR  Y ) SEN Z
( X OR  Y ) AND Z
( X OR  Y ) OR  Z
( X OR  Y ) NOT Z


  X NOT Y   NOT Z
( X NOT Y )     Z
( X NOT Y ) ADJ Z
( X NOT Y ) SEN Z
( X NOT Y ) AND Z
( X NOT Y ) OR  Z
( X NOT Y ) NOT Z
```

This stage also searches for the individual words so visual inspection may verify that the desired phrase is the proper intersection or union of the occurrence lists. Each conjunctive group contains a phrase where the intersection is the null set to demonstrate that false returns are eliminated.

The fourth stage contains 4-, 5-, and 6-word conjunctive phrases. These are chosen from the data files.

5.4 BROWSE DEMONSTRATION

The browse overlay produces the formatted fields and the text of the documents that contain the phrases specified in a previous search. The output of the search effort is retained by item number. The browse sections require specification only of the search phrase number. First, the formatted fields for the document are displayed, and the user is asked if a display of the text is also desired. The text of the document is then sent to the screen. After the display of each document, the user may go on to the next document or may terminate the browse of the phrase.

The browse demonstration uses a subset of the search section phrases and puts the text to output.

6.0 EVALUATION

The purpose of this section is to collect the results of previous sections with respect to evaluating the suitabilities of the various types of equipment considered for document retrieval and message handling applications. In addition, configuration and sizing guidelines for production systems are developed, and areas warranting further investigation are identified.

The demonstration system was configured with the A.U. developed on this project but used available AFP's, host computer, and disc storage system. The system configured for the demonstration (Figure 3-1) is not an optimal configuration for the document retrieval and message handling application, primarily due to the inadequacy of the disk subsystem which was available. Nevertheless, the configuration did provide an adequate base to develop an operational software environment in which the performances of the associative unit and array processors could be tested on crucial tasks.

6.1 Performance of the Equipments

An important goal of this project was to evaluate the use of associative and array processor architectures in advanced document retrieval and message handling systems. A list of the tasks central to both applications appears in Table 6-1. Associated with these tasks are the performances of the Associative Unit and Advanced Flexible Processor on these tasks.

TABLE 6-1 TASK ALLOCATION

Task	Equipment Type	Performance*	Notes
SEARCH			
Segment Table Lookup	AU	3.25 usec per search token	1
Segment Scanning	AFP	6.25 usec per search token	4
Occurrences Merge	AFP	4.35 usec per search operation	5
UPDATE			
Lexical Decomposition	AFP	3.212 usec per document token	2
Stopword Elimination	AU	3.25 usec per document token	1
Sort	AFP	6.292 usec per pass per document token	5
Indexing	AFP	$0.21\ T^{0.85}$ usec	
WORD COMPRESSION	AU	7.5 to 12 usec per document token	

* Assumes only one AFP.

Notes:

1) Based on the code used for stopword elimination with compressed tokens which was 1.625 usec per compressed token.
2) Based on the AFP code for lexical decomposition.
3) Based on AFP code for an 8-way merge on blocks of 512 compressed tokens which was 3.146 usec per pass per compressed document token. The total sort time on T uncompressed tokens is $6.292\ T \log_{512} T$ microseconds.
4) Assuming one AFP cycle per byte-length comparison and 25,000 byte segments. This task was not performed by an AFP in the demonstration system.
5) Assuming three AFP cycles per 6-byte occurrence pointer, and 5800 occurrence pointers per search operation.
6) Assuming 6 AFP cycles to unpack, compare, and repack each entry.

Examination of Table 6-1 reveals that the processing times for the central tasks of document retrieval are very short for the associative and array processors being evaluated. To obtain a clearer impression of these performances, Table 6-2 shows these performances applied to the large document retrieval system characterized in Tables 2-7 and 2-8, assuming 100 simultaneous users. As can be seen, the associative unit is virtually idle during searching (0.01% duty factor) and one AFP is adequate for nearly 40 times as many users that is 4000 users! The adequacy of a single AFP was also shown during the demonstration (Section 5). It is equally clear that a single disk unit is at least two and a half times too slow for even 100 users. It should be noted that text compression does not affect the search times if a constant dictionary segment size (25,000 bytes) is assumed.

In creating a temporary data base for updating, the performances of the associative unit, AFP, and disk are better matched. When profiling is considered, however, the disk performance again dominates the system. While a system performance of 5 hours to update still compares favorably with the current implementation for that retrieval system, an alternative approach such as that suggested in Section 2.6 should be investigated.

TABLE 6-2 TASK PERFORMANCES

Task	AU Time	AFP Time	Disk Time**
SEARCH			
Segment Table Lookup	78 us/sec	-	-
Segment Scanning	-	15 ms/sec	1.2 sec/sec
Occurrences Merge	-	11 ms/sec	1.25 sec/sec
Total	78 us/sec	26 ms/sec	2.5 sec/sec
UPDATE			
Lexical Decomposition	-	5.6 sec/mo	9.8 sec/mo
Stopword Elimination	5.7 sec/mo	-	9.8 sec/mo
Sort	-	33 sec/mo	58.6 sec/mo
Indexing	-	42.5 ms/mo	5.8 sec/mo
Profiling (10^5 commands)	1.1 sec/mo	328 sec/mo	4.7 hr/mo
Total	6.8 sec/mo	367 sec/mo	4.7 hr/mo
WORD COMPRESSION	15 sec/mo	-	-

* Assumes 100 simultaneous users entering one command each per 10 seconds with a data base of 350 million document tokens, and a monthly update of 1,750,000 tokens. It is also assumed that all AFP computations are processed by one AFP, and all storage is supported by one disk drive.

** Update disk time is based on one disk with 25,000 bytes per track, capable of sequentially reading or writing at an average rate of 1 track in 20 msec, and S_d = 12.2MBytes, S_v = 2MBytes, and S_o = 5.25 MBytes, where:

S_d = Storage for documents

S_v = Storage for vocabulary

S_o = Storage for occurrence lists

6.2 Configuring a Production System

In this section, the results of the analysis and the demonstration which were obtained during this project were applied to develop system configurations for production work in document retrieval and message handling. It is assumed that the system is located at a central facility and that the users interact with the system by remote time sharing terminals.

First, the demonstration showed very clearly that the general purpose computer (PDP 11/70) used here is inadequate for the tasks of managing and scheduling for a large number of simultaneous users as well as performing the substantial bookkeeping operations necessary.

Second, it is difficult to imagine a sytem requiring more than one AFP in the near future, particularly if a better profiling approach is developed, since one AFP can support thousands of concurrent users.

Initially, it appears unnecessary to include an associative unit in the system or to incorporate word compression. Instead, the tasks performed by the associative unit could be handled by the AFP. Using a binary search algorithm, the Segment Table Look-up and Stopword Elimination tasks in Table 6-2 would require about 0.4 msec/sec and 7.4 sec/month, respectively.

The area requiring the greatest departure from the analysis and demonstration systems is the disk memory subsystem. Clearly, the average disk access and transfer times must be reduced. This can be approached by a multiple disk system capable of concurrent transfers (multiple controllers and channels). The figures of Table 6-2 for disk times are based on accessing and transferring randomly selected tracks of 25,000 bytes in 50 msec each. This is an effective channel rate of 0.5 MByte

per second. Commercially available large-disk systems with 10,000 MBytes capacities and dual channel controllers should be able to deliver an effective channel rate of 6 MBytes per second. This reduces the disk times shown by a factor of 12, allowing concurrent searching by about 500 users and reducing the monthly update time to 25 minutes. Small high density disks of 500 MByte capacities are becoming available with effective channel rates of 0.6 MBytes per second for randomly selected full-track transfers. Employing twenty of these with an adequate number of controllers should provide a 10,000 MByte subsystem with a aggregate 12 MBytes/sec channel rate. This would support 1000 users and require a 15 minute monthly update.

The configuration developed in this section for the large document retrieval or message handling application is illustrated in Figure 6-1. Although an external cache memory is shown, the main memory of the general purpose machine might be used initially.

6.3 Topics for Further Study

During the work of this project, several topics warranting further investigation were identified.

6.3.1 Dictionary Reduction

This topic was addressed in Section 2.8.3. Significant improvement in system performance should be realized with morphological reduction of the dictionary and spelling error eliminations. This performance improvement should occur through higher computational efficiency and greater retrieval accuracy.

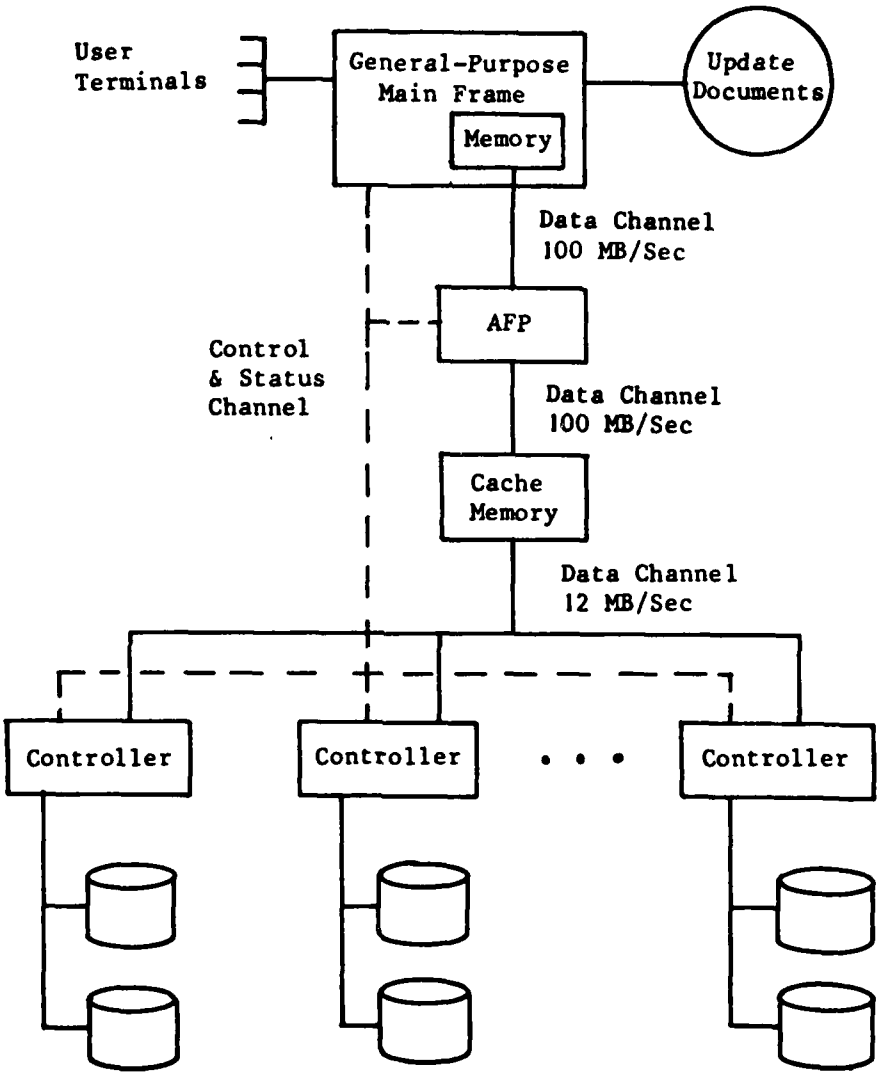


Figure 6-1. A Production System Configuration

6.3.2 Cache Memory Management

The search behavior of users should be examined further to determine the effectiveness of a cache memory, size requirements, and a cache management algorithm. One possibility which should be explored is the use of a thesaurus. When a search token is specified, associated tokens are looked up in a thesaurus, to anticipate tokens which might be used in subsequent commands. The dictionary segments for these associated tokens could be scanned during idle periods. Alternatively, the toke. ..ould be scanned for one user while a segment is being scanned for a t ..n specified by a different user.

Another approach might form a sub-dictionary for each ..er based on the mission profiles whenever the update process is executed. Then, when a user signs on, the appropriate sub-dictionary would be loaded into semiconductor memory. This might reduce the number of segment scans. In addition, each new token specified should be added to the sub-dictionary for that user.

6.3.3 Word Sense Discrimination

The search command syntax provides an awkward approximation to a natural language query format. For example, a query for "mission records of supersonic flights" might be specified by a command like "MISSION$1 SEN RECORD$ SEN SUPERSONIC SEN FLIGHT$3". In executing this command occurrences of RECORD and FLIGHT used as verbs would have to be processed along with the desired noun occurrences. This could be eliminated by dividing the occurrence lists according to the parts of speech as used in the documents. Bell Laboratories has reported a system called Writer's Workbench which claims an accuracy of 95% in assigning parts of speech in text. A study of the trade-off between added computation for word sense discrimination against search operation speed and retrieval quality should be undertaken.

6.3.4 A Table Look-Up Memory

Except for word compression, the primary use of the associative unit was for table look-up. With a 4-bit ALU in each of the 256 cells, the associative unit requires 2 memory cycles for a byte comparison and 48 cycles for a 24 byte look-up. A conventional memory of 256 24-Byte words and a 24-Byte comparator would require 8 memory cycles to perform an equivalent binary search and would require only 20% of the comparator logic. Thus, a micro-programmed binary search memory for segment look-up and stopword elimination could be designed to be 6 times faster and five times less expensive.

APPENDIX A

DEMONSTRATION DOCUMENT
RETRIEVAL SYSTEM

USERS MANUAL

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

A-1.0 INTRODUCTION

A-1.1 General Introduction

In order to demonstrate the document retrieval and document data base management capabilities of the AFP Array configured with associative memories, a command language, RCL (Retriever Command Language), has been defined. This language provides a means to build a data base around a collection of documents and to retrieve documents based on the words and their relative locations in the text of the documents. The purpose of this document is to describe these capabilities and their application.

A-1.2 Major Applications

There are four major applications which are handled by RCL. They are the following:

1. Creation and storage of data bases of documents.
2. Expansion (merge and purge) of data bases.
3. Retrieval of documents that satisfy specified criteria.
4. Scanning of retrieved document texts.

A-1.3 RCL Runs

In order to implement one of these applications, a period of time is needed which is devoted to the application. This period of time if called a run.

There are three distinct runs associated with RCL. An update run is devoted to creation, merging, renaming, and deletion of data bases. A search run is devoted to the retrieval of documents that satisfy specified criteria. Finally, a browse run is devoted to the scanning of the texts of retrieved documents.

A-1.4 RCL Files

Each RCL run requires basic input files and two of the runs (update and search) produces new files. The following are the basic files involved in RCL runs:

1. Document file on magnetic tape - Primary input for the data base creations and storage activity.
2. Data base files - Primary outputs of update runs and primary inputs for search activities.
3. Library file - Keeps data on the data bases available to search and browse.
4. Result list file - Primary output of a search operation and primary source identifying documents for browsing.

Table A-1 summarizes the information on the RCL files.

TABLE A-1. FILE SUMMARY CHART

FILE NAME	CONTENTS	RUNS USING FILE
DATA BASE FILES	A collection of files organized in a manner that supports document retrieval activities.	UPDATE, SEARCH, BROWSE
RESULT LIST FILE	A collection of occurrences that is the result of a query entry.	SEARCH, BROWSE
LIBRARY FILE	A file that stores the names of the available data bases and pertinent information on accessing the data base.	UPDATE, SEARCH

A-1.5 RCL Command Groupings

Besides search expressions, which are described in Section A-6.2, a user's entry can be either an RCL command or a response to a system prompt. A system prompt is provided whenever only user commands without parameters are required. RCL commands can be grouped in terms of runs for which they are exclusively available or in terms of being menu commands.

The UPDATE, SEARCH, BROWSE, LIBRARY, HELP and OUT commands are the menu commands. They support a user in executing the major runs by:

1. Providing a means to initialize the runs.
2. Tutoring on the usage of the RCL commands.
3. Providing information on the data bases available.
4. Allowing for the termination of RCL.

The CREATE, MERGE, OUT, REMOVE and RENAME commands are available during an update run. The CLEAR and OUT commands are available during a search run. There are no commands exclusively available for a browse run.

Instead, system. prompts are available for browse runs. Table A-2 summarizes the information on the RCL commands and their formats.

TABLE A-2. COMMAND SUMMARY CHART

COMMAND NAME	COMMAND FORMAT	USE
BROWSE	BROWSE,num	Browsing documents with occurrences in the specified result list.
CLEAR	CLEAR	Resets the current statement number to 0.
CREATE	CREATE,dbname,dflname	Creates a data base from a document file.
HELP	HELP,keyword	Provides tutorial information on usage of a command.
LIBRARY	LIBRARY	Provides pertinent information on the data base library.
MERGE	MERGE,dbname1,dbname2,dbname3	Merges two data bases.
OUT	OUT	Terminates an update run.

TABLE A-2. COMMAND SUMMARY CHART (CONT'D)

COMMAND NAME	COMMAND FORMAT	USE
REMOVE	REMOVE,dbname	Removes a data base from the library.
RENAME	RENAME,dbname1,dbname2	Renames a data base.
SEARCH	SEARCH,dbname	Initiates a search run.
UPDATE	UPDATE	Initializes an update run.

A-1.6 Mechanism for User Control

Five different types of inputs are required of a user of RCL. The RCL Program is a task that runs in the RSX-11M operating system of a PDP 11/70 host computer. Before executing the RCL task, the user must first become known to MAX, (Multiple Array Executive) the AFP executive by typing IEC and by replying with a password when requested. The second step is to reserve the AFP's by .REQ.

RCL is initiated by typing .RUN RCL. The menu commands are then displayed.

In order to create a data base, a file of documents to enter into the data base has to be created and be stored on magnetic tape. The document file is not created and be stored on magnetic tape. The document file is not created during RCL sessions but has to be done using available facilities on the PDP-11/70 host processor.

The course of an RCL session depends primarily on the RCL commands which the user enters. During a search run, the user's primary inputs are search expressions. These expressions specify the criteria for the retrieval of documents. During a browse run, the user can only make responses to the system prompts.

A-1.7 Description of Document

This document will describe in more detail the topics covered in this introduction. Section 3 describes the contents and formats of the RCL files. Section 4 described the operation and format of the RCL commands. Finally, Sections 5, 6 and 7 describe the entry organization and implementation events associated with the major applications. Section 5 deals with the creation and expansion of data bases, Section 6 deals with the retrieval of documents and Section 7 deals with the scanning of retrieved document texts.

A-2.0 APPLICABLE DOCUMENTS

The following documents may be referred to for further details:

77900500 Advanced Flexible Processor Microcode Cross Assembler

77900508 Advanced Flexible Processor Operating System

77900507 Advanced Flexible Processor Application Programmer

77900513 Advanced Flexible Processor Diagnostic

77900836 Associative Unit: Hardware Design Specification

Technical Memo Word Compression for Document retrieval

9199321

Final Report Applications of a Reconfigurable Array of Flexible Processor in Intelligence Information Retrieval.

A-3.0 RCL FILES

A-3.1 General Introduction

A major portion of RCL activity is centered around the creation and utilization of the various files. During an update run, document files can be used to create data base files and data base files can be merged to form data base files. For search runs, the data base files are used to access occurrence lists which are operated upon to produce result list files. Finally, for browse runs, the result list files provide document identifications of the documents the user should browse. These documents can be accessed from the data base files.

This section will describe the major files in more detail. It will describe the format, creation, utilization and aspects of each file under user control.

A-3.2 Document File

A document file is stored on magnetic tape in blocks of 5000 or less bytes per block and is delimited by an end of file mark. It is composed of a collection of documents. Each document has two sections. The formatted field section contains information about the document while the text section contains the text of the document. For each document, the formatted field section precedes the text section and a document is terminated by a formatted field record for another document or the end of file mark.

Each of the sections is comprised of a set of records. A record has a three byte header and a text portion which is made up of characters in the 64-character ASCII subset. The first two bytes of a record header indicates the record's length in bytes while the third byte indicates the record's type.

The formatted field section is made up of records of 7 types. The types are the document's ID, the document's title, the document's publication date, the names of the document's authors, the country which published the document and the institutions which supported the publication of the document. The formatted fields and their identifications are described in Table A-3. Figure A-1 has some sample formatted field records.

The text section is made up of records of one type. Those records are the document's text. The text is made up of sentences which are delimited by a period followed by two spaces.

A-3.3 Data Base File Organization

A data base is made up of three files - a dictionary file, an occurrence file and a text file as shown in Table A-4. A dictionary file has a segment table section and a dictionary section whereas a text file has a text index section and a text section.

A segment table record has a keyword field and a segment address field which identifies a track location in the dictionary file. There are at most 256 keywords and hence at most 256 segment table records. The dictionary section has a dictionary record for each term that occurs in the text section.* Each dictionary record has a term field, a number of occurrences field and an occurrence list address field.

*There are 256 most frequently occurring words that are not listed in the dictionary. These words are called "stop words."

TABLE A-3. DOCUMENT RECORD TYPES

RECORD TYPE	RECORD ID (OCTAL)	ORGANIZATION
Document I.D.	000	An 8 character document identification code is given.
Document Title	006	The title of the document is given.
Document Date	037	The date in which the document was published is given.
Document's Author	101	The names of the authors of the document are given.
Document's Country	103	The country where the document was published is given.
Document's Inst.	107	The names of the institutions supporting the publication of the document are given.
Document Text	143	Document text is given here.

TABLE A-4. COMPONENTS OF THE DATA BASE FILES

FILE NAME	SECTION	RECORD LENGTH	RECORD FIELDS NAME	USE
DICTIONARY	SEGMENT TABLE SECTION	Fixed	KEY	Segmentation Keyword Segment Track Address
	DICTIONARY SECTION	Fixed	TERM	N-Gram* String
			NOC	Number of Occurrences
			AD	Occurrence List Address
OCCURRENCE	OCCURRENCE SECTION	Varies	(DN,SN,WN)	(Document Number, Sentence Number, Word Number)
TEXT	TEXT INDEX SECTION	Fixed	TAD	Text Address
			FF	Formatted Fields
	TEXT SECTION	Varies	DOCUMENT	Text

*For purposes of data compression, the terms are packed into a sequence of n-grams. There are 256 n-grams where each n-gram is a sequence of alpha numeric characters.

LENGTH	TYPE	TEXT
13	0	79184662
145	6	AN INVESTIGATION OF THE LIFE OF TURBINE BLADES UNDER TERMINAL CYCLING AND VIBRATION IN A GAS FLOW
11	37	740214
31	101	ANATOLII ILLARIONOVICH
5	103	UR
55	107	UR SPECIAL DESIGN AND TECHNOLOGY BUREAU
51	107	UR INSTITUTE OF PROBLEMS OF STRENGTH
51	107	UR INSTITUTE OF PROBLEMS OF STRENGTH
61	107	UR ACADEMY OF SCIENCES OF THE UKRAINIAN SSR
15	143	EXTRACT.
21	143	DISSERTATION.
1771	143	THIS PAPER DESCRIBES ORIGINAL EXPERIMENTAL MEANS AND METHODS FOR DETERMINING THE LIFE OF GAS TURBINE BLADES UNDER CONDITIONS OF THERMAL CYCLING, VIBRATION, AND THE AGGRESSIVE ACTION OF A HIGH TEMPERATURE GAS FLOW. THE STRESS DEFORMED CONDITION OF THE TURBINE BLADES UNDER THESE CONDITIONS TAKING INTO ACCOUNT DESIGN FACTORS IS CONSIDERED. EXPERIMENTAL MEASUREMENTS AND THEORETICAL CALCULATIONS OF THE MOVEMENTS OF AN END SECTION OF A BLADE DURING ITS NONUNIFORM HEATING ARE GIVEN. THE ROLE OF ADDITIONAL THERMAL STRESSES ACTING ON TURBINE BLADES IN NONSTATIONARY CONDITIONS IS REVEALED. CURVES OF THE THERMOMECHANICAL FATIGUE OF TURBINE BLADES UNDER CONDITIONS OF THE PASSAGE OF A GAS FLOW ARE DETERMINED. THE EFFECT OF THE LEVEL OF VIBRATION STRESSES ON THE LIFE OF BLADES UNDER CONDITIONS OF THERMAL CYCLING AND VIBRATION IS ESTABLISHED. THE KINETICS AND CHARACTER OF THE DEVELOPMENT OF CRACKS IN BLADES OF EP220 ALLOY UNDER THE SIMULTANEOUS ACTION OF CYCLIC THERMAL CYCLING AND VIBRATION ARE INVESTIGATED.

Figure A-1. Sample Document File Records

The occurrence file is made up of occurrence list records. There is one occurrence list record for each term in the data base. Each entry in an occurrence list record identifies the location of an occurrence of the term in the texts of the data base. The components of an entry are the document number, the sentence number within the document and the word number within the sentence of the occurrence.

The text file has two sections. The text index section has a record for each document in the data base. Each record is composed of a pointer to the location of the text within the text file and the formatted field data for the test. The text section has the texts of all the documents in the data base.

The data base files are created during an update run. They are either created from a document file while executing a CREATE command or from two existing data bases while executing a MERGE command.

During a search run, terms in a search expression are referenced in the dictionary file and their occurrence lists are referenced from the occurrence file. During a browse run, the formatted fields and the text of a document are referenced from the text file. Figure A-2 demonstrates the interrelationship between the different sections by showing how one record in one section is used to access a record in another section.

The user is involved in the creation of a data base by first building document files. Then the user creates a data base for each document file by applying the CREATE command. Finally, he merges these data bases by applying the MERGE command.

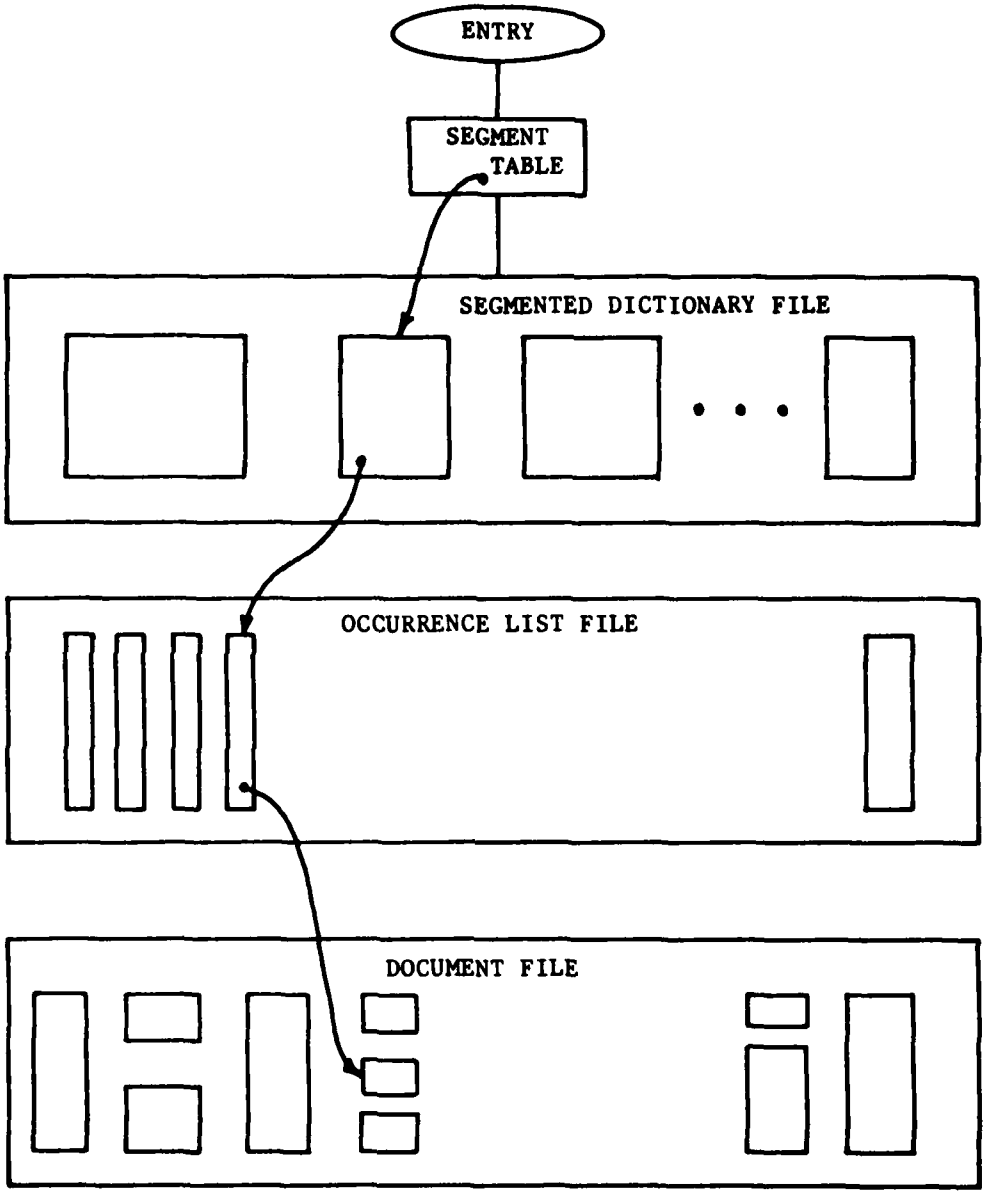


Figure A-2. Data Base Access Structure

A-3.4 Library File

The library file has a record for each data base that is available. Each record has the data base name, number of documents in the data base, number of text words in the data base, number of disk sectors the data base occupies, and location of the data base files on disk.

The library file is used to determine available data bases. When a LIBRARY command is given, data in the library file gets displayed. This feature provides the user information about the data bases.

The user is responsible for initially allocating space on the disk for the library file. Also, whenever a data base is created, removed or renamed during an update run, the contents of the library file are modified.

A-3.5 Result List File

A result list file is associated with each search query entered by the user. In addition to the statement number associated with a query, a result list file has the number of occurrences, the number of documents and the occurrence entries of all occurrences that satisfy the query expression.

A result list file is created during a search run each time a search expression is entered. A result list file remains available for search and browse applications until either a CLEAR command is entered during a search run, a SEARCH command is entered with a new data base name or the RCL session terminates.

The statement number for a result list file can be used as a term in a search expression. When this is done, the contents of the result list file are used in the processing of the search expression. Also, during browse runs, the documents identified in the result lists are those available to browse.

A-3.6 File Summary

Table A-1 summarizes the information presented in the files created and utilized during RCL runs.

A-4.0 RCL COMMANDS

A-4.1 General Introduction

RCL has three types of commands. They are the menu commands, update run commands and search run commands. The menu commands (BROWSE, HELP, LIBRARY, OFF, SEARCH and UPDATE) support RCL applications in the following ways:

1. Initiating RCL runs.
2. Terminating RCL sessions.
3. Tutoring users on the RCL commands
4. Providing information of the available data bases.

The update run and search run commands are available only during the corresponding runs.

This section described the formats and uses of the RCL commands. Table A-2 provides a summary of this information.

A-4.2 Command Formats

An RCL command is made up of a command key word and a list of parameters. The keyword must begin on column 1 of the line in which the command is being entered. A comma separates the keyword from the parameter list and parameters in the parameter list are separated from each other by commas. A blank represents the end of a command and comments can be provided after the first blank. Figure A-3 summarizes the format of an RCL command entry.

Keyword,p-list		
Keyword	-	Name of one of the RCL commands. A comma (or a blank if there are no parameters) terminates the keyword entry.
p-list	-	Parameters identifying input files, data bases, search statement number or command names. Some commands have no parameters. Parameters in the list are separated from each other by commas; embedded blanks cannot appear in the list. A blank terminates the p-list.

Figure A-3. Summary of RCL Command Entry Format

A-4.3 BROWSE Command

The BROWSE command is a menu command. It initiates a browse run. A BROWSE command requires that a statement number for a search expression be given as a parameter. After a BROWSE command is entered, the formatted field section of the first document listed in the occurrence list that is associated with the statement number is displayed along with a system prompt. The user can decide between having the text of the document displayed, the formatted field section for the next document displayed or the browse run terminated. Section 7 provides more information on browse runs and Figure A-4 summarizes the BROWSE command format.

BROWSE, num	
num	Statement number of the query whose results are to be browsed.

Figure A-4. BROWSE Command Format

A-4.4 CLEAR Command

The CLEAR command is available during serach runs. It resets the current statement number to 0 and purges the result lists of previous queries. There are no parameters associated with a CLEAR command. Figure A-5 summarizes the CLEAR command format.

```
CLEAR
```

Figure A-5. CLEAR Command Format

A-4.5 CREATE Command

The CREATE command is available during update runs. It creates a data base from a document file. The parameter required for a CREATE command is a name for the new date base. A legal data base name is made up of between 1 and 7 alphanumeric characters. Before the create command is entered, the document tape that has the documents to be incorporated into the data base must be mounted. After a data base has been created, it is listed in the library file and the data base is available to merge with other data bases or to search. Figure A-6 summarizes the CREATE command format.

```
CREATE,dbname,dflname

dbname          Name of the data base to be created.
```

Figure A-6. CREATE Command Format

A-4.6 HELP Command

The HELP command is a menu command. Information on an RCL command's format and usage is displayed after a HELP command is entered. A keyword for a command is required as a parameter. Figure A-7 summarizes the HELP command format.

HELP,keyword
keyword Keyword of the RCL command to review.

Figure A-7. HELP Command Format

A-4.7 LIBRARY Command

The LIBRARY command is a menu command. After a LIBRARY command is entered, information on the data base library is displayed. The information displayed includes the names of the available data bases and the number of documents and text words in each data base. Figure A-8 summarizes the LIBRARY command format.

LIBRARY

Figure A-8. LIBRARY Command Format

A-4.8 MERGE Command

The MERGE command is available during update runs. It creates a data base by merging two existing data bases. The command requires the names of the two existing data bases and a name for the new data base as parameters. A legal data base name is made up of between 1 and 7 alphanumeric characters. After the MERGE command has been entered, the new data base name gets listed in the library file and the data base is available for merging with another data base and searching. Figure A-9 summarizes the MERGE command format.

MERGE,dbname1,dbname2,dbname3	
dbname1	Data base name of one of the input data bases.
dbname2	Data base name of one of the input data bases.
dbname3	Data base name of one of the input data bases.

Figure A-9. MERGE Command Format

A-4.9 OUT Command

The OUT command is a menu command and is also available during update runs and search runs. After it has been entered as a menu command, the RCL session terminates. None of the RCL commands or results from search queries are available. After the command is entered during either update or search runs, the run terminates and the menu of commands is presented to the user. Figure A-10 summarizes the OUT command format.

OUT

Figure A-10. OUT Command Format

A-4.10 REMOVE Command

The REMOVE command is available during update runs. The command removes a data base from the library file. After the command has been entered, the data base is no longer available to merge with other data bases or to search. The name of a data base is required as a parameter for the REMOVE command. Figure A-11 summarizes the REMOVE command format.

REMOVE,dbname	
dbname	Data base name to remove the data base library.

Figure A-11. REMOVE Command Format

A-4.11 RENAME Command

The RENAME command is available during update runs. It changes the name of a data base in the library file. The name of the data base whose name is to change and the new data base name are required as parameters. A legal data base is made up of between 1 and 7 alphanumeric characters.

After the command is entered, future references to the data base will have to be made using the new name. Figure A-12 summarizes the RENAME command format.

RENAME,dbname1,dbname2	
dbname1	Old data base name.
dbname2	New data base name.

Figure A-12. RENAME Command Format

A-4.12 SEARCH Command

The SEARCH command is a menu command. It initializes a search run. A data base name specifies the data base to be searched. If a data base name is not specified, the previously specified data base will be searched. After the SEARCH command is entered, a statement number is displayed and the user can begin to enter search expressions. Section 6 provides more information on search runs and Figure A-13 summarizes the SEARCH command format.

SEARCH,dbname (optional)	
dbname	Data base name of data base to search.

Figure A-13. SEARCH Command Format

A-4.13 UPDATE Command

The UPDATE command is a menu command. It initializes an update run and allows the user access to the update commands. After the command is entered, the menu of update commands is presented. Section A-5 provides more information on update runs and Figure A-14 summarizes the UPDATE command format.

```
UPDATE
```

Figure A-14. UPDATE Command Format

A-5.0 UPDATE RUNS

A-5.1 General Introduction

The major application of update runs is the expansion of a data base. The update commands were developed primarily to support this application. The first step in expanding a data base is to create a data base around the documents to be added to the data base. After the new data base is created, it is merged with the original data base to form another new data base. The REMOVE command is then used to purge the two merged data bases and the RENAME command is used to retain for the resultant data base the original data base name.

A-5.2 Initialization

Update runs are initialized by selecting the UPDATE command when the menu of commands are available. If data bases are to be created, document files for the data bases have to be prepared. Section A-3.2 describes the organization of a document file.

A-5.3 Entry Organization

An update run begins when the UPDATE command is entered. An update run is terminated when the OUT command is entered. In order to expand a data base a document file must be created that includes the documents to add to the data base. This file must be created off line and should be mounted on tape drive 0 before initiating an update run. After the update run is initiated, the CREATE command is used to build a data base from the document file.

The MERGE command is used to create the expanded data base from the expanding data base and the newly created data base. The MERGE command requires that a name for the resultant data base be specified as a parameter. This name is usually a temporary name which will be changed once the name of the expanding data base becomes available.

The name of the expanding data base becomes available by first removing the expanding data base from the library file by using the REMOVE command. This releases the name of the expanding data base and the resultant data base can be assigned that name by applying the RENAME command. Unless the documents in the document file are to be used to expand another data base, the REMOVE command could be used to purge the newly created data base from the library file.

A-5.4 Implementation Events

After a CREATE or a MERGE command is entered, new data base files are created and a new entry is made in the library file. After the REMOVE command is entered, the data base files are released and an entry is removed from the library file. The RENAME command changes an entry in the library file. After the OUT command is entered, the menu commands are displayed and the update commands become inactive. Refer to Section A-8.0 for a sample update run.

A-6.0 SEARCH RUNS

A-6.1 General Introduction

The major application for search runs is to determine which documents in a data base meet certain criteria in terms of the words in their text and the relative positions of the words. The criteria are expressed by search expressions. Each search expression is associated with a statement number. The output for a search expression is a result list file which specifies the location of occurrences that satisfy the search expression.

A-6.2 Search Expressions

A search expression is composed of terms, operators and parenthesis where at least one blank separates the terms and operators. A term can either be a word, a truncated term, a string of words or a statement number associated with another search expression. Each term that is listed in the dictionary has an occurrence list associated with it. An occurrence list is stored in the occurrence file of the data base. A truncated term is a term that has a word beginning followed by an "$". Such a term refers to all words that have the same beginning. If the "$" is followed by a numeral, the numeral indicates the number of characters in addition to the word beginning that are allowed. For a truncated term, the occurrence list is derived from the union of the occurrence lists that has the same prefix as that specified in the truncated term and the specified number of characters. For a string of words, the occurrence list is derived from the occurrence lists for the words in the string. It identifies occurrences of the string of words. For statement numbers, the occurrence list is stored in a result list file. Since no result list files are available for statement numbers greater than or equal to the current statement number, any such number is treated as a word.

The search operators are binary operators in that they operate on two occurrence lists and produce another occurrence list as output. the set operators AND,OR (inclusive OR) and NOT are available. The other operators, ADJ and SEN, are called positional operators because they identify occurrences in the two lists that share a component in their text positions. The ADJ operator is used to identify words in the two lists that are in the same sentence and are adjacent while the SEN operator identify words that are in the same paragraph and sentence. Table A-5 describes these operators in set theoretical terms.

Search expressions are evaluated from left to right. Expressions inside parenthesis are evaluated first. The operations involving the SEN and ADJ operators are evaluated before the operations involving the OR, NOT and AND operators. The ADJ operator is evaluated before the SEN operator.

A-6.3 Initialization

A search run is initialized after the SEARCH command is entered when the menu commands are available. The current statement number is displayed and the user can begin to enter a search expression for that number. If the data base name gets changed from the previous search run, the current statement number is set to 1. Any statement number less than the current statement number can be used as a term in the search expression.

A-6.4 Entry Organization

A search run begins when the SEARCH command is selected from the menu commands and ends when the OUT command is entered. If the command is entered on any other position on the line, it will be interpreted as a term in a search expression. During a search run, the user can either enter a search expression or enter the CLEAR command to reset the current statement number to 0.

TABLE A-5. FORMULATION OF SEARCH OPERATIONS

SEARCH OPERATION	SET OPERATION	DEFINITION OF RESULT
OR	Union	R_3 = [(dn,sn,wn) | (dn,sn,wn) R_1 or (dn,sn,wn) R_2]
AND	Intersection	R_3 = [(dn,sn,wn) | ((dn,sn,wn) R_1 and (s_2)(w_2) ((dn,s_2,w_2) R_2) or ((dn,sn,wn) R_2 and (s_1)(w_1) ((dn,s_1, w_1) R_1))]
NOT	Difference	R_3 = [(dn,sn,wn) | (dn,sn,wn) R_1 and (s_2)(w_2) ((dn,s_2,w_2) R_2)]
ADJ	Intersection	R_3 = [(dn,sn,wn + 1) | (dn,sn,wn) R_1 and (dr,sn,wn + 1) R_2]
SEN	Intersection	R_3 = [(dn,sn,o) | (w_1) (w_2) ((dn,sn,w_1) R_1 and (dn,sn,w_2) R_2]

TABLE A-5. FORMULATION OF SEARCH OPERATIONS

SEARCH OPERATION	SET OPERATION	DEFINITION OF RESULT
R_1	occurrence list of the left search term	dn - document number
R_2	occurrence list of the right search term	sn,s_1,s_2 - sentence number
R_3	result occurrence list	wn,w_1,w_2, - word number
	(dn,sn,wn) - document id	

A-6.5 Implementation Events

Whenever a search expression is entered, the terms for the search expression are identified and occurrence lists for the terms are gathered. The operators in the expression are applied on the occurrence lists to produce the result list for the expression. The result list is stored in a result list file and is identified by the current statement number. The file can then be referenced by other search expressions and browse runs by referring to its statement number. Refer to Section A-8.0 for a sample search run.

A-7.0 BROWSE RUNS

A-7.1 General Introduction

During a browse run, the user can examine the documents identified in a result list file. The displaying of a document is divided into two parts. First, the formatted fields of a document are displayed. They provide information about the source and the text of the document. Table A-3 describes the various formatted fields.

Second, the user can choose between examining the text portion of the document or the formatted field portion of the next document. Thus the user can quickly browse through the formatted fields of the documents until a document of particular interest is encountered and then examine the text for that document.

A-7.2 System Prompts

Since the activities during browse runs do not require the specification of parameters, system prompts (where the user is presented a list of choices and indicates his or her choice by depressing a key on the keyboard) can be used instead of commands to direct the course of a browse run. The choices provided to a user always include the choice of paging to the next document and exiting from the browse run.

After a page of text is displayed or the formatted fields for text are displayed, the following prompt appears:

```
ENTER      N = NEXT DOCUMENT   T = TEXT          E = EXIT BROWSE
```

If the user depresses the "N" key, the formatted field portion of the next document is displayed. If the user depresses the "T" key, the next page of text is displayed. Finally, if the user depresses the "E" key, the browse run terminates.

A-7.3 Initialization

A browse run is initialized by entering the BROWSE command and the statement number of a result list file. After the BROWSE command is entered, the formatted field portion of the first document identified in the result list file is displayed.

A-7.4 Entry Organization

A browse run begins when the BROWSE command is entered and terminates either when the "E" key is depressed, after the "N" key is depressed and the last available document is displayed, or after the user's key entry when the last page of the last document is displayed. During a browse run, the only entries available to a user are those described by the system prompt.

A-7.5 Implementation Events

The system responses made to user entries are based on the user's choices to the system prompts. If the user selects the next document option, the formatted field portion of the next document is displayed as well as the following prompt:

```
ENTER      N = NEXT DOCUMENT   T = TEXT          E = EXIT BROWSE
```

If the text option is selected, a page of text and the prompt gets displayed. Finally, if the exit browse option is selected, the message "BROWSE COMPLETE" and the menu commands are displayed. Refer to Section A-8.0 for a sample browse run.

A-8.0 RCL SAMPLE TERMINAL SESSIONS

I. UPDATE SESSION

```
 RUN RCL
SELECT MODE - ..UPDATE .. SEARCH .. BROWSE .. LIBRARY .. HELP .. OUT
UPDATE
SELECT COMMAND - .. CREATE .. MERGE .. REMOVE .. RENAME .. OUT
CREATE,TEMP,XFL
TEMP CREATED
SELECT COMMAND - .. CREATE .. MERGE .. REMOVE .. RENAME .. OUT
MERGE,TEMP,CIIO,SAVE
MERGE COMPLETE - SAVE CREATED
SELECT COMMAND -  .. CREATE .. MERGE .. REMOVE .. RENAME .. OUT
REMOVE,TEMP
TEMP REMOVED
SELECT COMMAND - .. CREATE .. MERGE .. REMOVE .. RENAME .. OUT
REMOVE,CIIO
CIIO REMOVED
SELECT COMMAND - .. CREATE .. MERGE .. REMOVE .. RENAME .. OUT
RENAME,SAVE,CIIO
CIIO CREATED, SAVE REMOVED
SELECT COMMAND - .. CREATE .. MERGE .. REMOVE .. RENAME .. OUT
OUT
EXIT UPDATE MODE
SELECT MODE - .. UPDATE .. SEARCH .. BROWSE .. LIBRARY .. HELP .. OUT
LIBRARY
```

DATA BASE LIBRARY

DATA BASE	NUMBER DOCUMENTS	NUMBER WORDS
C110	1051827	71385429
CIRC	830772	58780631
XLAX	192320	13442018
COLC	121949	8612742
UPDT	38821	2757840

SELECT MODE - .. UPDATE .. SEARCH .. BROWSE .. LIBRARY .. HELP .. OUT

OUT

II. QUERY AND BROWSE SESSION

RUN RCL

SELECT MODE - .. UPDATE .. SEARCH .. BROWSE .. LIBRARY .. HELP .. OUT

SEARCH,CI10

SEARCH MODE - ENTER QUERY AFTER STATEMENT NUMBER

01

SECURE COMMUNICATIONS

SECURE	176 OCCURRENCES	152 DOCUMENTS
COMMUNICATIONS	937 OCCURRENCES	438 DOCUMENTS
RESULT	5 OCCURRENCES	4 DOCUMENTS

02

ENCRYPTION$1 OR CRYPTO

ENCRYPTION	4 OCCURRENCES	2 DOCUMENTS
ENCRYPTIONS	13 OCCURRENCES	7 DOCUMENTS
CRYPTO	2 OCCURRENCES	2 DOCUMENTS
RESULT	19 OCCURRENCES	10 DOCUMENTS

```
03
SATELLITE COMMUNICATIONS
SATELLITE                    842 OCCURRENCES    518 DOCUMENTS
COMMUNICATIONS               937 OCCURRENCES    438 DOCUMENTS
RESULT                       602 OCCURRENCES    375 DOCUMENTS
04
(1 AND 3) OR (2 AND 3)
1                              5 OCCURRENCES      2 DOCUMENTS
3                            602 OCCURRENCES    375 DOCUMENTS
2                             19 OCCURRENCES     10 DOCUMENTS
3                            602 OCCURRENCES    375 DOCUMENTS
RESULT                         2 OCCURRENCES      2 DOCUMENTS
05
OUT
EXIT SEARCH MODE
SELECT MODE-..UPDATE..SEARCH..BROWSE..LIBRARY..HELP..OUT
BROWSE,4
DOCUMENT = 1 OF 2 NUMBER OF WORDS = 563
INSTITUTION:
     NAVRB
DATE:
     1973
COUNTRY:
     UK
TITLE:
     BRISTOL THE ELECTRONIC WARSHIP
ENTER N = NEXT DOCUMENT  T = TEXT  E = EXIT BROWSE
```

T

EXTRACT OF REPORT

VITAL RADAR, WEAPONS AND COMMUNICATIONS SYSTEMS FROM GEO. MARCONI ELECTRONICS, TOTALLING MORE THAN 3M, ARE CARRIED BY THE BRISTOL, THE ...

ENTER N = NEXT DOCUMENT E = EXIT BROWSE

N

DOCUMENT = 2 of 2 NUMBER OF WORDS = 598

DATE:

1973

COUNTRY:

UK

TITLE:

COMMUNICATION SECURITY

ENTER N = NEXT DOCUMENT T = TEXT E = EXIT BROWSE

E

BROWSE COMPLETE

SELECT MODE - .. UPDATE .. SEARCH .. BROWSE .. LIBRARY .. HELP .. OUT

OUT

A-9.0 SYSTEM MESSAGES

System Messages are provided in response to user entries. They are designated either to provide the user feedback on his or her entry or to describe the options currently available to the user. There are three types of messages. An error message (E) indicates that an error was detected in the user's entry. An informative message (I) informs the user that the system has completed the procedure that he or she specified and what were the results of the procedure. Finally, a prompt (P) informs the user on the available entry options. Table A-6 summarizes these messages.

TABLE A-6. SYSTEM MESSAGES

MESSAGE	TYPE	SIGNIFICANCE	ACTION
COMMAND NAME NOT RECOGNIZED	E	The HELP command cannot determine which RCL requires tutorial information.	Select an RCL command
DATA BASE NAME ALREADY USED	E	The new data base name cannot be used for its intended purpose because the name is already assigned to another data base.	Select a different data base name
DATA BASE NAME NOT LEGAL	E	The new data base name does not satisfy the criteria for a data base name.	Change one of the data base names
DATA BASE NOT FOUND IN LIBRARY	E	The data base selected to search or for update activities is not listed in the library.	Use the LIBRARY command to determine which data bases are available
db CREATED	I	A data base named "db" has been created and is available to search and to merge with other data bases.	None
db CREATE, db REMOVED	I	The data base that had the second data base name has been renamed first data base name.	None

TABLE A-6. SYSTEM MESSAGES (CONT'D)

MESSAGE	TYPE	SIGNIFICANCE	ACTION
ENTER N = NEXT DOCUMENT	P	These are the entry options available to the operator.	Depress either the "N" key or the "E" key
ENTER N = NEXT DOCUMENT T = TEXT E = EXIT BROWSE	P	These are the entry options available to the operator.	Depress either the "N" key, the "T" key, or the "E" key
EXIT SEARCH MODE	I	The search mode commands and capabilities are no longer available.	None
EXIT UPDATE MODE	I	The update mode commands are no longer available.	None
ILLEGAL CHARACTER IN QUERY	E	An illegal character for a search query has been detected.	Change query
ILLEGAL COMMAND	E	Command entered does not correspond to one of the commands listed on the menu.	Enter one of the menu commands listed
ILLEGAL NUMERAL	E	The numeric parameter entered is not a numeral or is not in the proper range.	Enter a legal numeral for the numeric parameter

TABLE A-6. SYSTEM MESSAGES (CONT'D)

MESSAGE	TYPE	SIGNIFICANCE	ACTION
MERGE COMPLETE db CREATED	I	The merge operation has been completed and a data base name "db" is available to search and merge with other data bases.	None
n	I/P	The current statement number will be assigned to the next query entered. A query can be entered after a statement number is displayed.	Enter a search query, the CLEAR command or an OUT command.
n OCCURRENCES n DOCUMENTS	I	Indicates number of occurrences and documents that satisfy the term.	None
QUERY NOT LEGAL	E	There is something wrong with the syntax of the query just entered.	Determine the syntax error and enter a legal query
RCL	I	An RCL session has commenced. RCL commands are available.	None
RESULT LISTS CLEARED	I	CLEAR command has been completed. The current statement number has been reset to 1 and a result lists cleared.	None

TABLE A-6. SYSTEM MESSAGES (CONT'D)

MESSAGE	TYPE	SIGNIFICANCE	ACTION
RESULT n OCCURRENCES n DOCUMENTS	I	Indicates the number of occurrences and documents that satisfy the search query.	None
SEARCH MODE - ENTER QUERY AFTER STATEMENT NUMBER	I	Search mode has been initialized and search queries can now be entered.	None
SELECT MODE - .. CREATE .. MERGE .. REMOVE ..RENAME.. OUT	P	These are the Menu commands available to the user.	Select one of the commands listed
SELECT MODE - .. UPDATE .. SEARCH .. BROWSE .. LIBRARY .. HELP .. OFF	P	These are the commands available during an update run.	Select one of the commands listed
STOP WORD. NOT LISTED IN DICTIONARY	I	The term entered is a stop word and is not listed in the *dictionary*.	None

A-10.0 GLOSSARY

BROWSE	An RCL run that provides the user a means to review the retrieved documents. The BROWSE command is a menu command that initializes browse runs.
COMMAND	Instructions given by the user to direct the course of an RCL session and to specify the files involved in RCL operations.
DATA BASE	A collection of files that are organized to support document retrieval activities. The document texts available for retrieval are stored in text file of the data base.
DOCUMENT	The major data structure for the document retrieval system. A document has two parts: a formatted field section that contains information about the document and a text section that contains the document's text.
DOCUMENT RETRIEVAL	The process of retrieving a subset of documents from a data base that satisfy a specified criteria.
MENU	A list of commands that are available to the user at a particular time. After a menu is presented, the system waits until the user enters one of the commands listed in the menu.

MERGE	The process of combining two data bases to form a new data base that has all of the documents of both data bases and is organized to support the retrieval of documents in the new data base. MERGE is also a command available during update runs.
OCCURRENCE	The identification of an instance of an occurrence of a word in a text of a document. The components of an occurrence's identification includes the document number, paragraph number, sentence number and word number.
PROMPT	System messages to the user during browse runs which defines the available choices. The user's response is limited to depressing one key.
RCL	Retriever Command Language
RESULT LIST	A list of word occurrences that satisfy specified criteria. Each identification of an occurrence contains an identification of the document that has the occurrence. Hence, the relevant documents are identified in result lists.
RUNS	A period of time that is devoted to a specific application.
SEARCH	An RCL run that determines which documents satisfy specified criteria. The SEARCH command is a menu command that initializes search runs.

SEARCH EXPRESSION	The format in which the criteria for document retrieval activities are specified.
UPDATE	An RCL command that provides the user a means to build new data bases. An UPDATE command is a menu command that initializes update runs.

APPENDIX B

THE ADVANCED FLEXIBLE PROCESSOR, ARRAY ARCHITECTURE

The Advanced Flexible Processor (AFP) is a relatively powerful computer employing a highly parallel architecture. It has been designed to stand alone, hosted by a general-purpose computer, or to function within arrays of Advanced Flexible Processors. All of the features for efficient interprocessor communication and control are built into each Advanced Flexible Processor to allow efficient computation in a multiprocessing environment. The Advanced Flexible Processor was developed by an advanced computer research division of Control Data called the Information Sciences Division (ISD). The Information Sciences Division began work on the Advanced Flexible Processor in 1976. Our primary goal was to develop a programmable computing machine that would provide the computational power and speed required by many of the intense algorithmic processes associated with image processing, while providing some of the flexibility of a general-purpose machine.(1)

Early Research and Development in Multiprocessing

In 1968 Control Data began research into the feasibility of performing some of the tasks associated with image processing functions, such as modular change detection on digital computers. CDC began this effort by testing algorithms on the super computer of that day, the CDC 6600. Based on this preliminary algorithmic study effort on the modular change detection problem, ISD began development of the 1280 Change Detection System which was a hardwired-logic implementation build specifically to perform the Change Detection Algorithm. Our experience in the designing and building of special-purpose hardwired systems for image processing applications indicated the need for a more flexible approach to the development of special-purpose systems.

In 1972, the Information Sciences Division began development of the Flexible Processor (FP), which was a programmable, special-purpose computer employing a highly parallel architecture. The Flexible

Processor, like its successor the Advanced Flexible Processor, was designed to operate as an individual programmable processing element in an array of other individually programmable elements. The Flexible Processor used a global bus interconnection system between processors. Later investigations began to determine other interconnection network architectures which might prove to be more optimally suited for a Multiple Instruction, Multiple Data Stream (MIMD) type of array architecture(2,3). The products of this initial research into various interconnection schemes resulted in ISD developing a ring connected architectural approach to linking Flexible Processors in large multiprocessing arrays.

In 1976 Control Data delivered its first modular change detection system built around the Flexible Processor ring connected architecture to Wright-Patterson Air Force Base. Research indicated that a processor capable of performing at computational rates 10 times that of the Flexible Processor was in order and would be required to meet the burgeoning computational demands of the 1980's (4-7). Thus, Control Data Corporation began the development of the Advanced Flexible Processor using the latest LSI technology which was developed by CDC for use in its most advanced Cyber computers.

AFP Hardware Overview

An Advanced Flexible Processor is implemented on four large scale integrated (LSI) circuit panels. The component technology is emmitter coupled logic (ECL) chips. Each LSI panel carries a total of approximately 500 F200K ECL logic chips and 1,100 ECL 100K logic chips. The Advanced Flexible Processor employs the same freon cooling system used in CDC's Cyber 200 series computers. This technology provides an increased reliability figure at the chip level of approximately 100 times that achievable using ECL 100K logic chips in an air-cooled environment.

The rough computational capabilities provided by an array of 16 Advanced Flexible Processors would be approximately 3.2 billion arithmetic and logical operations per second. A far larger number of operations could be added to the total if one were to count the many operations associated with operand transfer and data management which are concurrently performed by the AFP in support of the arithmetic and logical computations.

Interconnection Technology. AFP systems employ a ring connected architectural concept. The interprocessor communication between two adjacent Advanced Flexible Processors in the communications ring is approximately 800 million bits per second. A unique characteristic of the ring connected architecture employed by the Advanced Flexible Processor provides a distinct advantage in the performance capability of multiprocessor systems. Program partitioning strategies allow one to realize proportional increases in available ring system intercommunication bandwidth as processors are added to the multiprocessor array. This feature is in direct contrast to other multiprocessor architectures in which interprocessor communication is strangulated as processors are added to the system. As a result of this unique feature, an array of 16 Advanced Flexible Processors may provide overall system bandwidth for intercommunications of 26 billion bits per second.

AFP Performance Benefits. Comparisons between the performance of the Advanced Flexible Processor and other current super computers have been made on the image processing Change Detection Algorithm. The Advanced Flexible Processor has been determined to be approximately 2,000 times faster than a CDC 6600 on the Change Detection Algorithm, and to provide approximately 100 times the capability of the CDC 7600 computer. The Advanced Flexible Processor is found to perform 20 times faster than its

predecessor, the Flexible Processor. In terms of cost effectiveness, the Advanced Flexible Processor appears to be at least two orders of magnitude more cost-effective than it predecessor, the Flexible Processor.

Architectural Concepts

The following discussion will review some of the issues related to the choice of a multiprocessing solution for those problems for which general-purpose uniprocessors do not provide adequate solutions.

Pipelined Processing. Consider a processing facility composed of a single processor to which is presented an incoming stream of data elements. Computations are to be performed upon these incoming data elements, and output results are to be provided on a real or near real time basis (Figure B-1a). When the number of operations to be performed on the incoming data elements increases to the point where a single processor cannot provide output results within the required real or near real time constraints, or where an input backlog is steadily growing, then it would be required that the compute power of the single processor be augmented by the addition of processors into the system to work jointly on the common task at hand.

The common task would be partitioned among the added processors in a pipeline fashion where each processor would operate only upon a single serial stage of the entire computation, and would pass its intermediate results onto the next cooperative processing element, which would be working on the next sequential stage of the computation (Figure B-1b). As each computational stage completes the processing of a data element, the next data element in sequence may be input to that stage, and stage processing initiated. The pipeline is "full" when there is a data

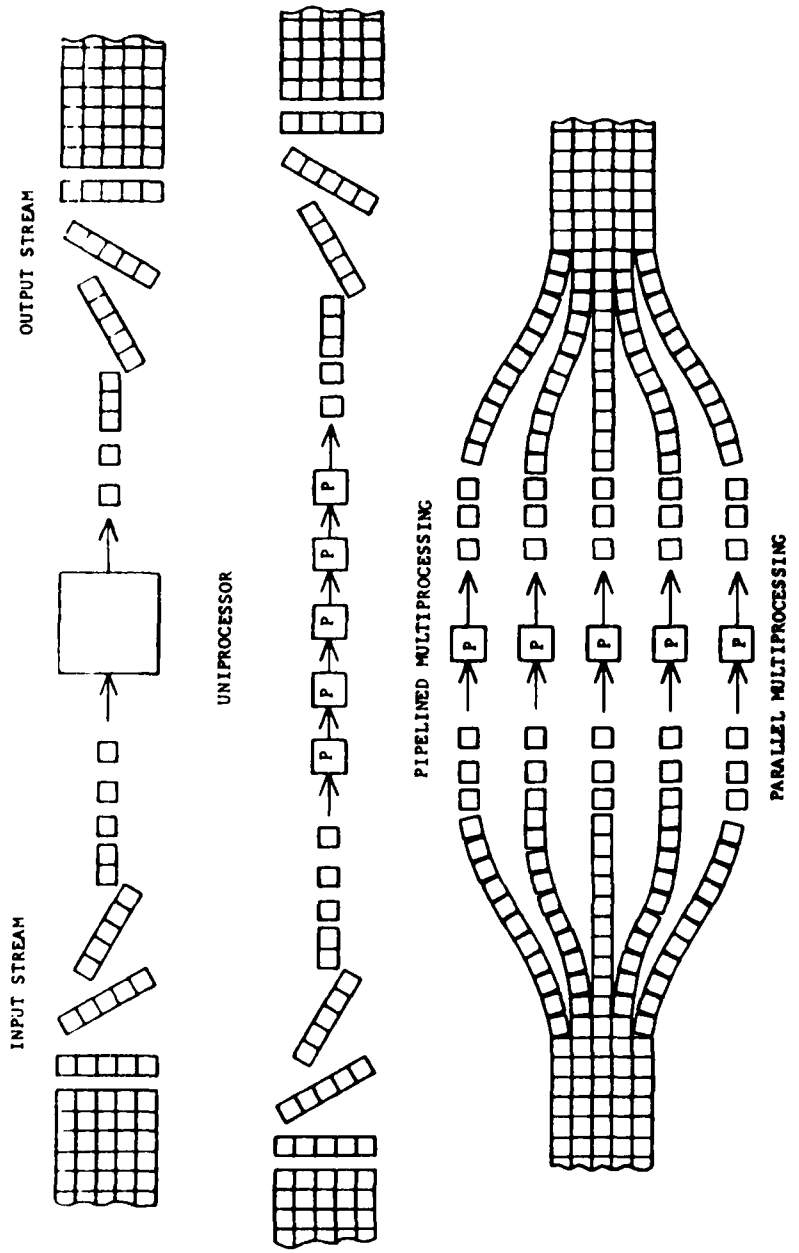


Figure B-1. Pipelined and Parallel Processing

element simultaneously being processed through each and every stage of the pipeline. Each of the N processors, corresponding to the N stages of the pipeline, would then be busy, contributing to the total processing power brought to bear on the problem.

Parallel Processing. If the incoming data rate of the proposed system were to increase to a point beyond the individual I/O capability of a single processor, then it would be required that processors be added to the computation in a parallel fashion, each performing identical operations on a parallel set of data elements (Figure B-1c). In summary, one can state that when the number of instructions in a particular algorithm increases beyond the capability of a single processor required processors would be added in a pipeline fashion, whereas when the I/O rate of an individual processor is exceeded, then processing elements are required to be added in a parallel fashion. The Modular Change Detection system developed by the Information Sciences Division consisted of four pipelines with ten processors in each pipe.

General Multiprocessor Taxonomies

In general it is not adequate to simply provide capability for only parallel or pipeline configurations of processing elements, or for that matter some parallel-pipelined combination thereof. Algorithms are generally more complex than that, and require more complex feedback paths, such as exemplified in recursive types of algorithms.

Four generalized types of intercommunications architecture for multiprocessing that may be considered are shown in Figure B-2 and are: 1) a global bus interconnection architecture where all communication between the processors occurs on a single, common data bus; 2) a fully interconnected architectural scheme where each cooperating processing element has a unique data path to every other processing element in the

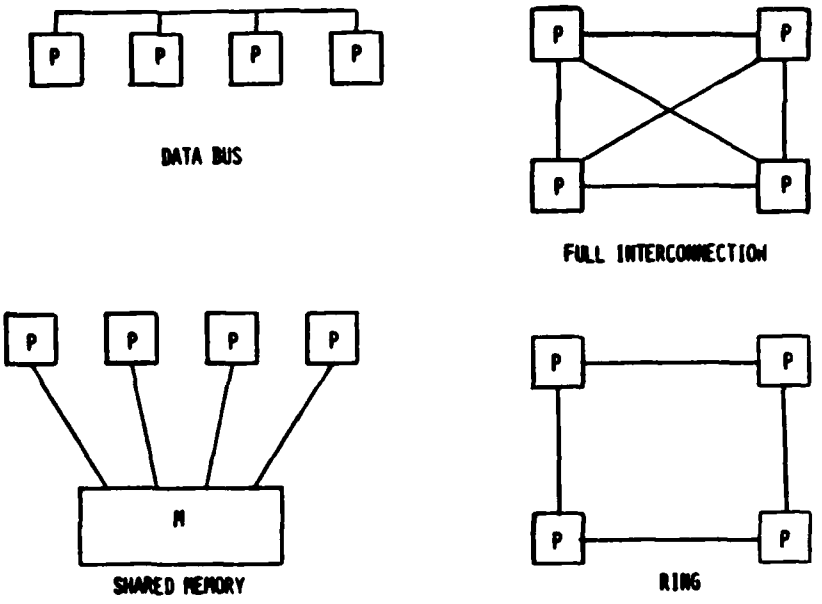


Figure B-2. Interprocessor Communication System

array; 3) a shared memory type of processing element interconnection where all communication and data transfer occurs through a common, shared memory facility; and 4) a ring connected architectural concept which consists of a circular interconnection of processing elements, where each processing element is directly connected to its two neighboring processors in the ring. The Advanced Flexible Processor uses the latter two interconnection schemes. The Advanced Flexible Processor uses both a dual, counter-rotating ring interconnection system, as well as a common shared memory facility.

Each of the previously mentioned interconnection architectures possess characteristic strengths and weaknesses, requiring evaluation on the basis of several criteria. These architectures may be evaluated on the basis of: 1) system reliability, 2) expansion capability, and 3) cost.

System Interconnect Reliability. From the standpoint of reliability, the shared memory system and the global bus both have problems in the area of single-point failures. If a failure of the bus or the central memory occurs, the entire system is incapacitated. A ring system, when bypass hardware is employed, demonstrates very good fault tolerant characteristics.

The fully interconnected system is the best of four systems considered in the area of fault tolerance since each processor has a dedicated path to every other processor for intercommunications.

System Interconnect Expandability. From the standpoint of expansion limitations, the shared memory system has problems in that the number of ports are fixed. Expanders can be used to alleviate this problem to some degree, but physical construction problems are utlimately met. Also, the memory bandwidth of the shared memory system is fixed and is relatively slow, thus limiting the degree of practical expansion.

A global bus system has limited fanout capabilities; electrical problems are generally encountered after a relatively low number of processors are added to the system. Also, the global bus system demonstrates the lowest bandwidth capability of all of the systems, since all of the processing elements used the common shared bus. In fact, the operating bandwidth of the global bus system will never reach its theoretical maximum due to the idle time spent while processors access the bus, release the bus, and resolve bus access conflicts.

The fully interconnected system is limited in that the number of ports are fixed with respect to the processing elements. While expanders can be used, ultimately physical problems will be encountered.

In the ring system, the bandwidth between adjacent processors is fixed; however, utilizing the special characteristics of a ring connected architecture provides system intercommunication bandwidths which tend towards the arithmetic product of this fixed interprocedure bandwidth and the number of processors in the system. Thus, a proportionate increase in supporting intercommunications bandwidth is available as processors are added to the system. Expansion within a ring connected system is, of course, virtually unlimited and has a very low cost impact.

System Interconnect Cost Performance. One may obtain a measure of the cost effectiveness of the four generalized architectures by plotting the cost to throughput ratio for each architecture as a function of the number of processors in the system (Figure B-3).

The shared memory system is the most expensive of the four generalized architectures, with the global bus system coming in at close second. The fully interconnected system is about 5 times more cost-effective than a global bus approach for a 30-processor system; however, the ring system

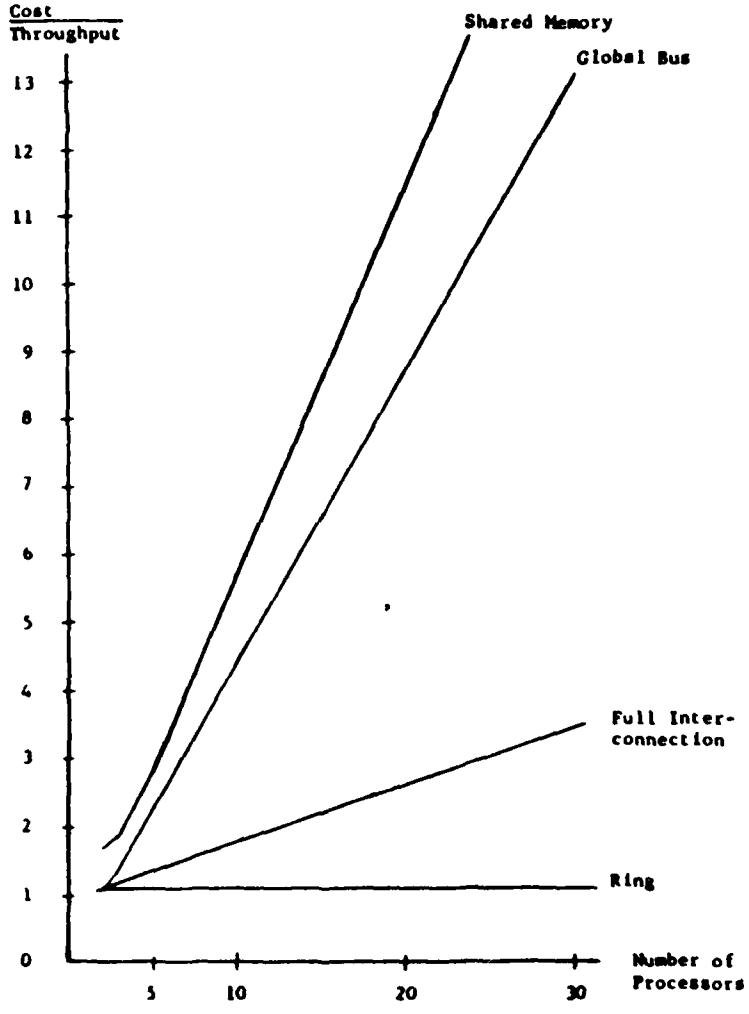


Figure B-3. Normalized System Cost/Throughput vs. Number of Processors

is superior to all when the process is partitioned to take advantage of the unique bandwidth characteristics that a ring connected architecture provides.

Advanced Flexible Processor Architecture

The Advanced Flexible Processor is a unique and powerful architecture providing an extremely high degree of flexibility and cost-effectiveness. It consists of 16 relatively autonomous functional units interconnected by a 16 x 18 port, crossbar interconnect. Each of the data paths interconnected by the crossbar is 16 bits wide. Table B-1 describes the functional unit breakdown of the Advanced Flexible Processor. A conceptual functional organization of the AFP is shown in Figure B-4.

Computations may be streamed through the Advanced Flexible Processor very efficiently due to dual I/O port characteristics of the internal architecture. Data elements may be independently streamed in and out of the Advanced Flexible Processor through any one or all of the four I/O channels. For example, data may be streamed in through one of the memory I/O channels, computations performed, and then streamed out through one of the other three I/O channels simultaneously.

Multifunctional Parallelism. The internal architecture of the Advanced Flexible Processor allows multiple computational streams to be constructed and executed in parallel. By way of example, one might imagine the multiply unit requesting operands from one of the memory I/O ports and one of the data memories, while at the same time an adder may be requesting the product computed by the multiplier on a previous machine cycle and another data element from one of the remaining three data memories to serve as input operands for an addition operation. At the same time, the remaining adder may be using the sum produced by the

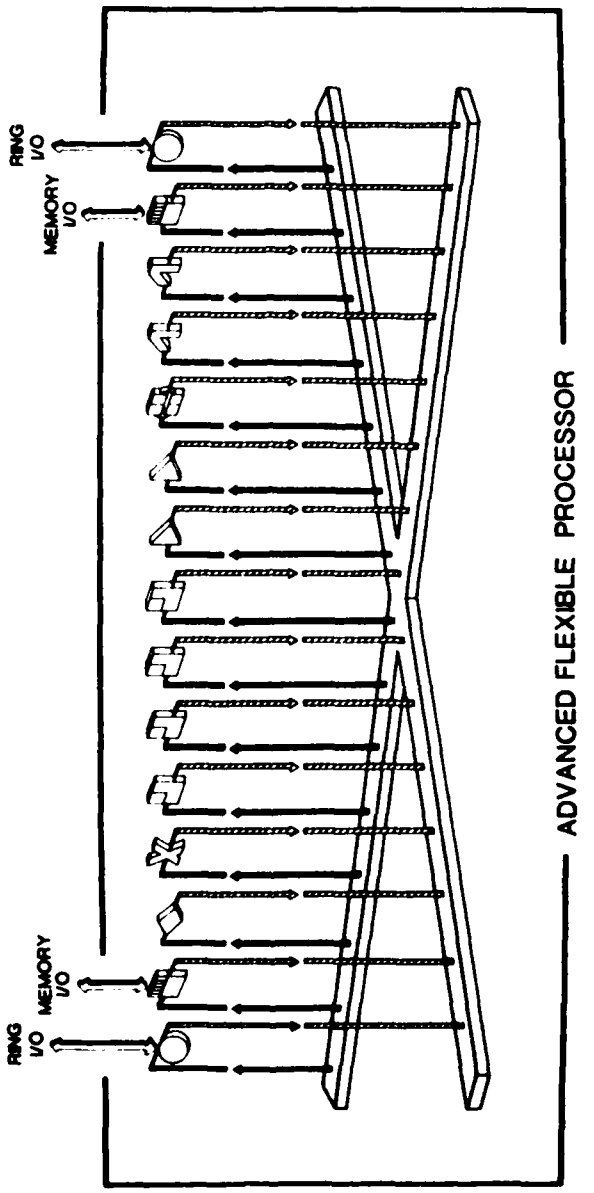


Figure B-4. Functional Organization of the Advanced Flexible Processor

TABLE B-1. FUNCTIONAL UNIT BREAKDOWN

Number of Units	Type of Functional Unit	Number of Pipelined Segments
2	External Memory Access Unit	1
2	Ring Port I/O Units	1
1	Control Unit	2
2	Adders Unit	2
1	Multiplier Unit	3
2	Shift Boolean/Logic Unit	2
4	2K Data Memory Units	2
2	8 Word File Registers	2

first adder on a previous machine cycle and the result from a shift boolean operation to perform a subtraction. The ultimate goal, of course, is to get as many functional units executing simultaneously as possible and thereby achieving the highest concurrency of execution.

Each of the internal functional units of the AFP are I/O buffered to their respective crossbar ports as shown in Figure B-5. Each functional unit is equipped with input latch registers, buffering the crossbar inputs, and output latch registers, buffering the functional unit outputs to the crossbar. This design allows the intermediate storage of variables between the functional units and thus allows the functional units of the AFP to be pipelined together with the maximum flexibility. Single or multiple pipelined chains are easily supported through the crossbar as a result of this method of "direct data hand-off" between the functional units.

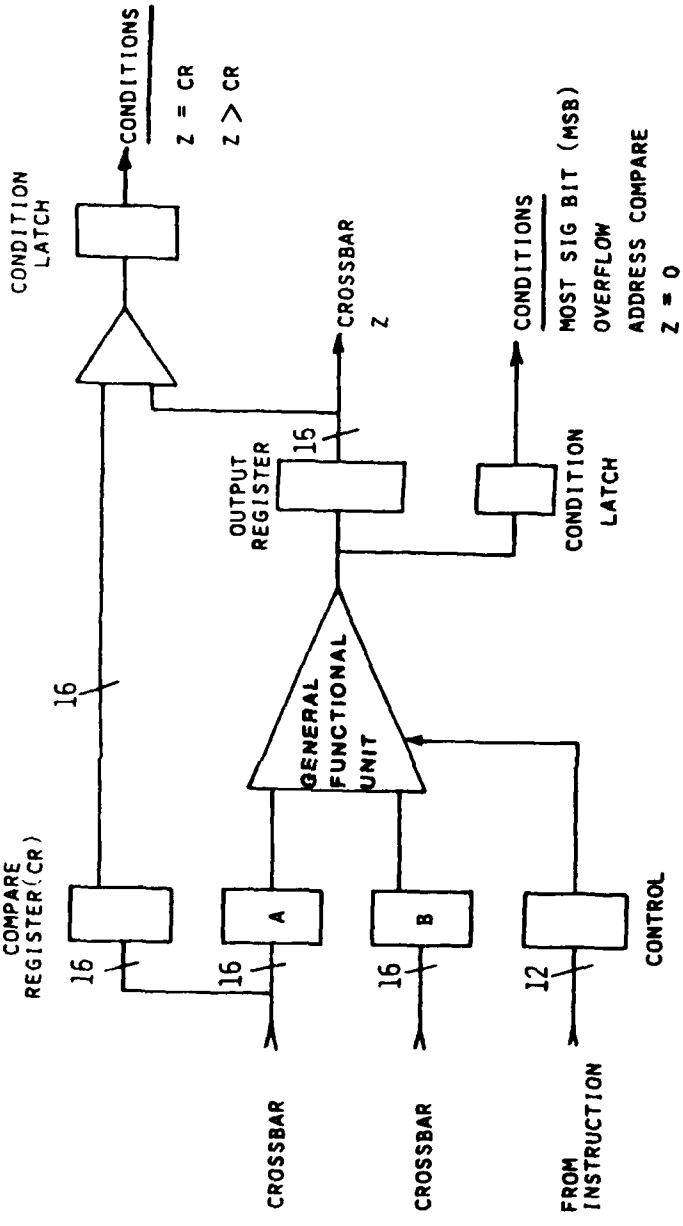


Figure B-5. Register Level Organization of a Generic AFP Functional Unit

Advanced Flexible Processor Performance

The machine cycle time of the Advanced Flexible Processor is 20 nanoseconds. Every functional unit can provide results every 20 nanoseconds. Thus, 50 million 16-bit multiplies, 200 million 16-bit data memory references, and 100 million 16-bit adds or subtracts, etc. can be performed every second. The maximum operational speed of the Advanced Flexible Processor, therefore, is 800 million operations per second when all 16 functional units are executing.

AFP I/O Performance. The ring port I/O unit provides the interface for each Advanced Flexible Processor to the ring interconnect system. Two ring ports are provided to each Advanced Flexible Processor and thus the capability for dual-ring interconnection systems exists. The ring port I/O unit handles all of the data management, synchronization, and protocol required to communicate on the ring system without interrupting the arithmetic processing of the Advanced Flexible Processor.

The external memory access unit provides the interface between the AFP and the central, high-performance, random access memory store. Each external memory access unit can provide peak memory store. Each external memory access unit can provide peak data I/O rates of 3.2 billion bits per second and sustained I/O rates of 800 million bits per second. Thus, the total sustained capability of an Advanced Flexible Processor from the two ring port I/O units and the two external memory access units is 3.2 billion bits per second.

AFP Computational Performance. The multiply unit of the Advanced Flexible Processor provides the capability to produce two 16-bit products or one 32-bit product every 20 nanosecond machine cycle. The multiplier also provides the capability to do population and significant counts. The two adders provide the capability of performing four 8-bit adds, two

16-bit adds, or one 32-bit add every 20 nanosecond machine cycle. The shift boolean units allow barrel shifts of up to 15 bits performed every 20 nanosecond machine cycle and is capable of performing all of the 16 basic boolean logic functions. Each data memory allows the reading or writing of one 16-bit word every machine cycle. The file memories allow the reading and writing of four 16-bit words every machine cycle. The control unit manages program execution and handles branching and accessing of programming instructions.

The individual program memory within the control unit of each AFP consists of 1,024 program instructions. Each program instruction is 200 bits wide and provides the capability of issuing 39 instruction parcels every 20 nanoseconds. The control bandwidth of the AFP is thus very high, and allows flexibility in control for the easy management and execution of the 16 functional units and the crossbar reconfiguration on a machine cycle basis. As a result, the Advanced Flexible Processor is capable of performing 100 million, 250 million, or 500 million arithmetic and logic operations every second in the 32-bit, 16-bit, or 8-bit modes of operation respectively.

Comparison Testing. Latching registers as shown in Figure B-5 are also provided within each functional unit for the storage of input comprand values. Arithmetic computations and testing of resultant outputs can thus be concurrently performed within all of the functional units. The current conditional status of each functional unit can therefore be provided to the control unit every machine cycle for branch decision processing. When counting all of the arithmetic and logic operations plus all of the comparison results provided by each of the functional units, one finds the Advanced Flexible Processor capable of performing an astounding 2.9 billion, 8-bit operations per second. This compute capability represents the upper theoretical limit for the Advanced Flexible Processor.

On average, a typical computational process can keep four of the arithmetic functional units plus several memory and I/O units busy concurrently, allowing a single AFP to achieve an average computational rate of about 200 to 250 million 16-bit arithmetic and logical operations per second.

The features provided by a single AFP are summarized in Table B-2. The very modular construction of AFP systems and of the AFP itself allows for very cost-effective system implementation. The modularization of functional units about the crossbar interconnect allows the enhancement of AFP performance specification by replacing existing functional units with specialized functional units designed specifically to meet performance requirements. Typical examples of specialized function are: fast fourier transform units, floating point add, multiple, and divide/square root units.

AFP System Architecture

System arrays of Advanced Flexible Processors are linked together and synchronized via facilities provided by the ring port functional units. Data elements 16 bits wide, along with 12 bits of control information, are passed between ring ports on adjacent AFP's. The control information to define the single processor or subset of system processors to which the message is to be sent.

Information identifying the appropriate data register file in which the incoming data element is to be stored is also contained within the control field of the ring packet. Each data memory is capable of defining 16 independent data files. Designated bits within the control field provide interprocessor synchronization information as well. Facilities within the ring port provide the logic capabilities to use these designated bits to achieve cross file synchronization. These

TABLE B-2. SINGLE AFP FEATURES

FEATURE	ADVANTAGE
250 MILLION ARITHMETIC COMPUTATIONS PER SECOND FOR EACH AFP	EXPANDABLE COMPUTE POWER TO MATCH APPLICATION
FUNCTIONALLY DESIGNATED INTERMEDIATE OPERAND REGISTERS	ALLOWS UNINTERRUPTED COMPUTATION STREAMING, ELIMINATING REGISTER RESERVATION HICCUPS
DIRECT DATA HAND-OFF BETWEEN 16 FUNCTIONAL UNITS THROUGH CROSSBAR SWITCH	PROVIDES BROADEST CAPABILITY FOR MULTIPLE CHAINING WITH NO REQUIREMENTS ON OPERAND INDERPENDENCE
DATA FAN OUT OF 1:16 ON ALL FUNCTIONAL UNITS	ELIMINATES OPERAND CONTENTION, ALLOWING MULTIPLE USE OF A SINGLE OPERAND IN ONE MACHINE CYCLE
FOUR INDEPENDENT DATA MEMORIES PROVIDING CONCURRENT ACCESS AND COMBINED CAPABILITY TO SUPPLY 16 INPUT REQUESTS SIMULTANEOUSLY	PROVIDES 8 KB OF CIRCULATING VECTOR STORAGE, AVOIDING COSTLY VECTOR LENGTH START-UP TIMES

TABLE B-2. SINGLE AFP FEATURES (CONT'D)

FEATURE	ADVANTAGE
FOUR INDEPENDENT I/O PORTS PROVIDING SIMULTANEOUS READ/WRITE ACCESS TO HPR MEMORY	ELIMINATES VECTOR LENGTH HICCUPS IN COMPUTATION STREAM PEAK BANDWIDTH 8 BILLION BITS/SECOND SUSTAINED BANDWIDTH 3.2 BILLION BITS/SECOND
200 BIT WIDE INSTRUCTION PACKET	INSTRUCTION ISSUE RATE IS 39 INSTRUCTION PARCELS/CYCLE OR 2 BILLION INSTRUCTIONS PER SECOND
TRANSPARENT SINGLE LEVEL INTERRUPT EXCHANGE MANAGEMENT	NO SPECIAL INTERRUPT EXCHANGE SOFTWARE PACKAGES REQUIRED
INSTRUCTION CACHE SIZE OF 1024 INSTRUCTION PACKETS, EACH 200 BITS WIDE	40 THOUSAND INSTRUCTION PARCELS PER PROGRAM INTERVAL

features assure that a processor is not capable of beginning a computational task until the appropriate single data file or set of data files which are to be used as operands in the pending computation are stored away in the processor.

These synchronizing control features also prevent another processor from over-writing files within a computing processor. Input and output FIFO buffering provides elasticity in communication between processors on the ring systems to minimize processor idle time. Thus, due to the built in capabilities of the ring port functional units, the processing elements are released from the inflexible lock-step synchronization required of other single instruction, multiple data stream (SIMD) machines and multiple instruction, multiple data stream (MIMD) machines. Further, the system allows for the construction of multiple elastic pipelines to be created across system AFP's, which function as powerful processing elements in the dual ring connected architecture.

A Minimum AFP System. A minimum AFP system configuration would consist of a host computer, presently a PDP 11/70, communicating with a single AFP via a modified ring port interconnection, MRP/C (Figure 6). An AFP operating as an attached processor in this configuration would enhance system performance of the host processor by providing a capability of 250 million additional arithmetic computations per second. The ring port interface units through which which rings of AFP's may be interconnected are indicated in Figure B-6 by the abbreviation RP( ).

Multiprocessor AFP Systems AFP's can be easily added to the minimum system shown in Figure B-6. A typical multiprocessor expansion is shown in Figure B-7. AFP's are interconnected on the host ring with each additional AFP augmenting the computational capabilities of the system by 250 million arithmetic operations per second. An additional

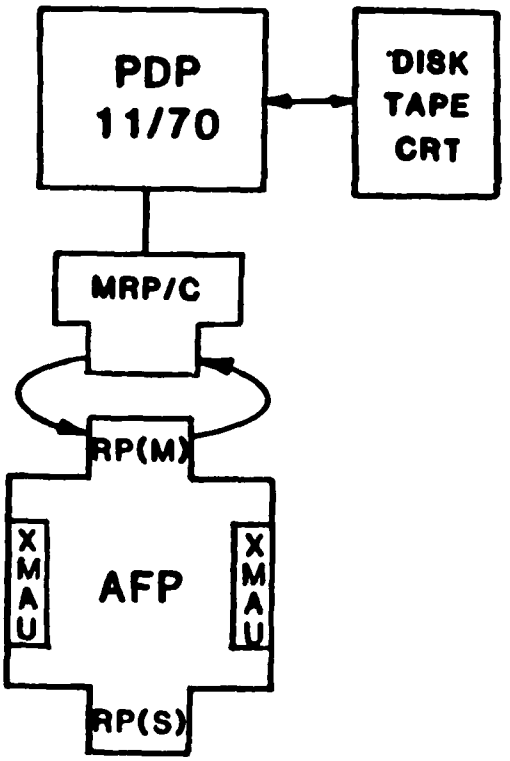


Figure B-6. Minimum AFP System Configuration

ring interconnection channel, shown in Figure B-7, is also provided for interprocessor communication and control. Up to 256 Advanced Flexible Processors can be supported on each system ring.

Centralized High Performance Memory. A multiprocessor system of AFPs may share a common, high-performance random access memory store (HPR) between processors. All system HPR requests sent from the external memory access units (XMAU) of the AFP's are managed by the Storage Access Controller (SAC). Multiple SAC's may be employed as memory requirements are expanded. Each SAC is capable of transferring data to and from the AFP array at a sustained rate of 6.4 billion bits per second.

This centralized, high-performance memory store may be expanded from 125 kilobytes to 16 million bytes, providing a maximum memory bandwidth of 12.8 billion bytes per second. The advanced technique of processor intercommunications significantly reduces processor idle time. Processor idle time is further reduced through a sophisticated hierarchical approach to mass memory and I/O management, which ensures continuous data support to the processing elements and a continuous computational flow. All memory and communication paths are designed to support extremely high bandwidths.

Centralized Mass Memory Facility. In addition to the high-performance central memory, AFP systems may be configured to provide a centralized mass memory facility composed of slower, relatively inexpensive memory technologies that can be accessed by each system AFP. The mass memory hierarchy can be configured to meet individual requirements and may include MOS random-access storage, disks, tapes, and memory archives, as well as high-speed interfaces to general peripheral I/O devices and display stations.

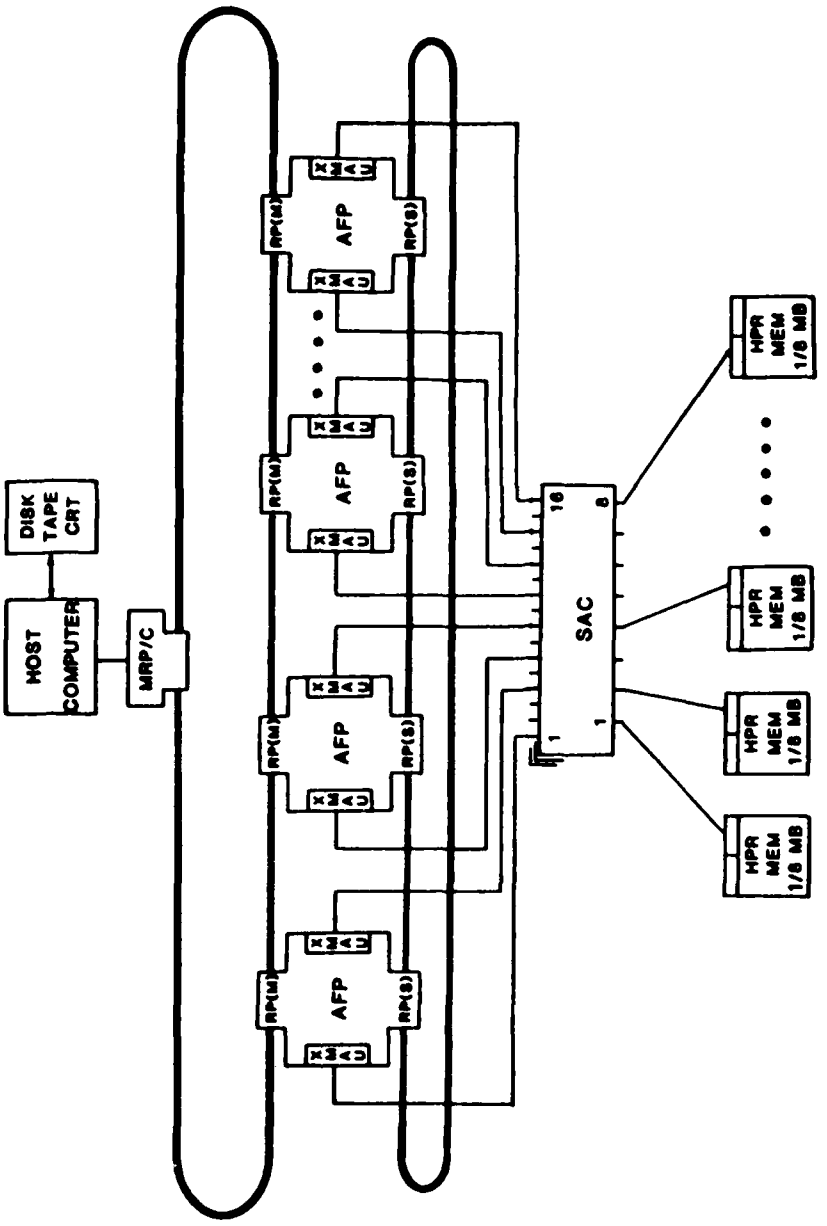


Figure B-7. Typical AFP System Configuration Showing Capabilities for Expansion

The MOS Memory technology provides low-cost random access storage at a fraction of a cent per bit. This cost compares to that of the higher performance technology used in the HPR central memory of 3-4 cents per bit. Sustained read/write data rates to and from MOS memory can exceed 1,000 million bits per second. MOS capacity may be expanded from 256 kilobytes to 1 billion bytes to provide ample random access storage at a low cost per bit ratio to cost-effectively meet the storage requirements for very large problems.

AFP System Performance for Specific Applications

A number of specific applications for the AFP have been studied at Informations Sciences Division. The performance of single and multiprocessor systems of AFP's has been assessed for these applications. The computational performance of the Advanced Flexible Processor on a representative set of these algorithms is shown in Table B-3. Beyond these areas of investigation there are yet broader applications for the Advanced Flexible Processor that are being investigated. Data retrieval system(8) as well as floating point applications are starting to be addressed.

AFP Software Facilities

The programming of the AFP system is presently done through the use of two very powerful software development tools. The first of these tools is the AFP cross assembler, MICA. The second tool is the AFP instruction level simulator, ECHOS. The MICA cross assembler and the ECHOS instruction level simulator allows all programming to be done "off-line." AFP programs can be written using the editor facilities of either a PDP 11/70 or a Control Data Cyber 700 series computer. The edited files are then processed by the MICA cross assembler. MICA checks for all illegal lexical and syntax usages as well as illegal hardware

TABLE B-3. AFP APPLICATION PERFORMANCE

APPLICATION	NUMBER OF AFP's	KERNAL RATE	TOTAL TIME
COMPLEX FFT, 1024 POINT	1	80 ns/BUTTERFLY	0.4 msec
16 BIT ACCURACY	4	20 ns/BUTTERFLY	0.1 msec
GEOLOCATION, 100,000 MESSAGES 100 LOCATIONS OF INTEREST	1	40 ns/PAIR	0.4 sec 10 MILLION COMPARES
2-DIMENSIONAL MATRIX DECONVOLUTION (55 x 80) ELEMENTS	1	20 ns PER MULTIPLY-ADD	6.6 msec
MULTISPECTRAL CLASSIFICATION (128 x 128) POINTS	16	480 ns/POINT COMPUTATION	8 msec
		1240 OPERATIONS PER POINT	2.58 BILLION OPERATIONS/SEC
MATRIX INVERSE (50 x 50) POINTS	1	2 usec/POINT	5.0 msec
MATRIX TRANSPOSE (32 x 64) ARRAY	1	10 nsec/POINT	20 usec
(1024 x 1024)		20 nsec/POINT	21 msec

usages. Functional unit and cross bar usage conflicts are identified to the programmer through the facilities of MICA. MICA produces a binary file of the submitted program which runs directly on the Advanced Flexible Processor.

The binary file produced by MICA also runs directly on ECHOS the AFP instruction simulator. ECHOS provides a register level simulation of the submitted program. ECHOS interactively executes the program in software precisely the way the program will run in the Advanced Flexible Processor. A programmer can single step through his program specifying the print out of all or a selected set of functional registers in the AFP. The accuracy, power, and detail of the ECHOS simulator allows a programmer to confidently expect his program to run the very first time it is run on an AFP. Thus, programming activities can be carried out with no interruption to useful AFP system data processing.

Development of higher level programming languages for the AFP is currently underway. FORTRAN, ADA, and the data flow language VAL which has been developed at the Massachusetts Institute of Technology and the Lawrence Livermore National Laboratory(9) are all candidates to be supported.

Summary

The Advanced Flexible Processor is a unique entry into the multiprocessing field. It provides the dynamic capabilities offered by an MIMD machine with advanced features provided by the interprocessor ring communications network; efficient utilization of the system processors is therefore effected. Within each Advanced Flexible Processor, dynamic multiple chaining can be achieved due to the superior flexibility of the intra-processor crossbar. Multiple functional units can be executed simultaneously with each functional unit providing a

broad range of instruction defined operational capabilities. Functional unit make up within an Advanced Flexible Processor can be varied and optimized for variable data sets. Multiple comparisons are available within each machine cycle for simultaneous multiple condition sensing.

Special-purpose functional units can replace existing functional units within the Advanced Flexible Processor, allowing processor capabilities to be tailored to the precise allowing processor capabilities to be tailored to the precise application requirements. Modular system construction allows compute power modularity; thus, processing systems can be cost-effectively tailored to the users individual requirements.

Future Computational Trends. The trends in computational requirements over the last 25 years has increased logarithmically by an order of magnitude every 8 years. Users will continue to demand higher performance, computational facilities at a rate matching or surpassing that of the previous 25 years. The demand appears to be insatiable as long as cost effectivity can be sustained. Semiconductor technology has been able to meet these demands over the past 25 years, however, presently there appears to be a slowing in the rate of technological advances within the semiconductor area. A circuit density increase by a factor of two every year as predicted by Moore's law is not presently being met, due to the increased problems in semiconductor fabrication that are being confronted.

There is little evidence that this trend will reverse itself over the coming years. The current trend indicated by the widening of this technological gap, seems to indicate that the only way to meet the computational requirements of the scientific community in the mid 1980's is through the application of multiprocessor technology. Development of this multiprocessor technology will provide the foundation to bridge the computational gap between 1985 and 1990. Future advances

in semiconductor technologies will yield increases in computational speed at the circuit level. These semiconductor advances will certainly be incorporated into multiprocessing hardware, and thus the skills we develop to employ multiprocessing in the 1980's will provide the stepping stones to meet the computational demands of the 1990's.

Acknowledgements

The information presented in this paper is a direct result of the collective knowledge gained by the personnel in the Information Sciences Division of Control Data. This knowledge has been gained from more than nine years of experience in the field of multiprocessing.

Literature Cited

1. Allen, G.R., A Reconfigurable Architecture for Arrays of Microprogrammable Processors, Special Computer Architectures for Pattern Processing, Purdue University, West Layfayette, Inc., CRC Publishing Corporation, to be published in 1981.

2. Hsu, T.T., On the Performance and Cost Effectiveness of Some Multiprocessor Systems, 1977 International Conference on Parallel Processing, August 1977.

3. Stenshoel, C.R., Production Image Processing System Design Study, Control Data Corporation Final Report to Centre National d'Etudes Spatiales, May 1977.

4. Juetten, P.G. and Allen G.R., An Image Processor Architecture, Control Data Corporation, Minneapolis, Minnesota, 1977.

5. Allen, G.R. and Juetten, P.G., SPARC - Symbolic Processing Algorithm Research Computer, /Proc. Image Understanding Workshop/, Science Applications, Inc., Report No. SAI-79-814-WA, 1978.

6. Allen, G.R., Advanced Image Processing systems Design Studies, Control Data Corporation Final Report to the Rome Air Development Center, Contract No. F30602-76-C-0362, March 1978.

7. Cyre, W.R., Allen, G.R., and Juetten, P.G., Symbolic Processing Algorithm Research Computer Progress Report, Control Data Corporation, Minneapolis, Minnesota, 1978.

Literature Cited (Cont'd)

8. Cyre, W.R., Applications of a Reconfigurable Array of Flexible Processors in Intelligence Information Retrieval, Control Data Corporation Final Report to the Rome Air Development Center, Contract No. F30602-78-C-0065, July 1979.

9. Ackerman, W.B., and Dennis J.B., VAL-A Value-Oriented Algorithmic Language: Preliminary Reference Manual, Laboratory for Computer Science, Massachusetts Institute of Technology.

APPENDIX C

ASSOCIATIVE UNIT

C-1.0 INTRODUCTION

The Associative Unit consists of 256 associative processing cells and a microprogrammable controller organized as illustrated in figure C-1.

The Associative Unit (AU) is designed to operate as an add-on unit for the Advanced Flexible Processor (AFP). The interconnections required between an Associative Unit and an AFP are summarized in Table C-1. The AU requires a 64-bit data and control input path, an 18-bit data output path, and a number of other control signal lines. The AU signals can be changed once in every AU machine cycle. The functions of the signals defined in Table C-1 are described further in section C-5.

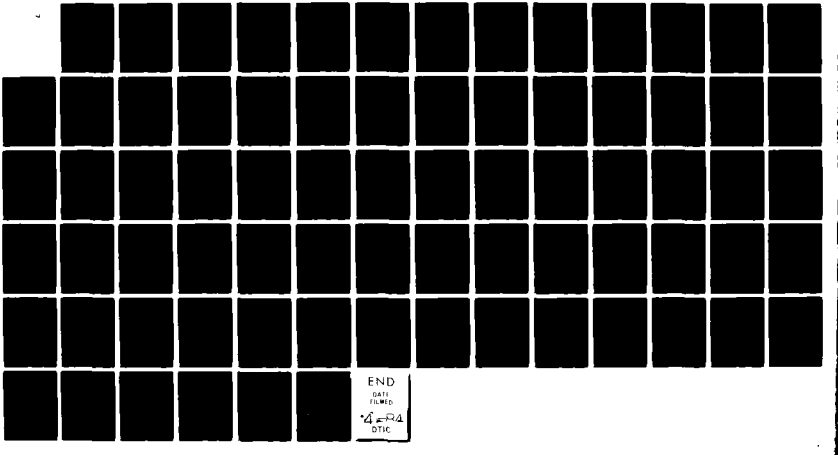
The internal machine cycle of the Associative Unit has a period of 120-125 nsec. and is synchronized to the 20 nsec. clock of the AFP. The Associative Unit is constructed with Schottky TTL logic and level shifters to accommodate the ECL signals of the AFP.

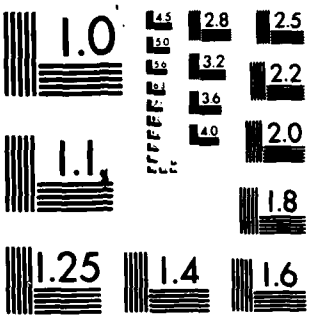
C-1.1 Bit Numbering Convention

The bit numbering in the Associative Unit begins with zero on the right (LSB) and increases to the left. The bit numbering in the AFP begins with zero on the left (MSB) and increases to the right.

AD-A139 385 ADVANCED DOCUMENT RETRIEVAL SYSTEM(U) CONTROL DATA CORP 3/3
MINNEAPOLIS MN W CYRE ET AL. JUL 83 RADC-TR-83-168
F30602-79-C-0231

UNCLASSIFIED F/G 9/2 NL





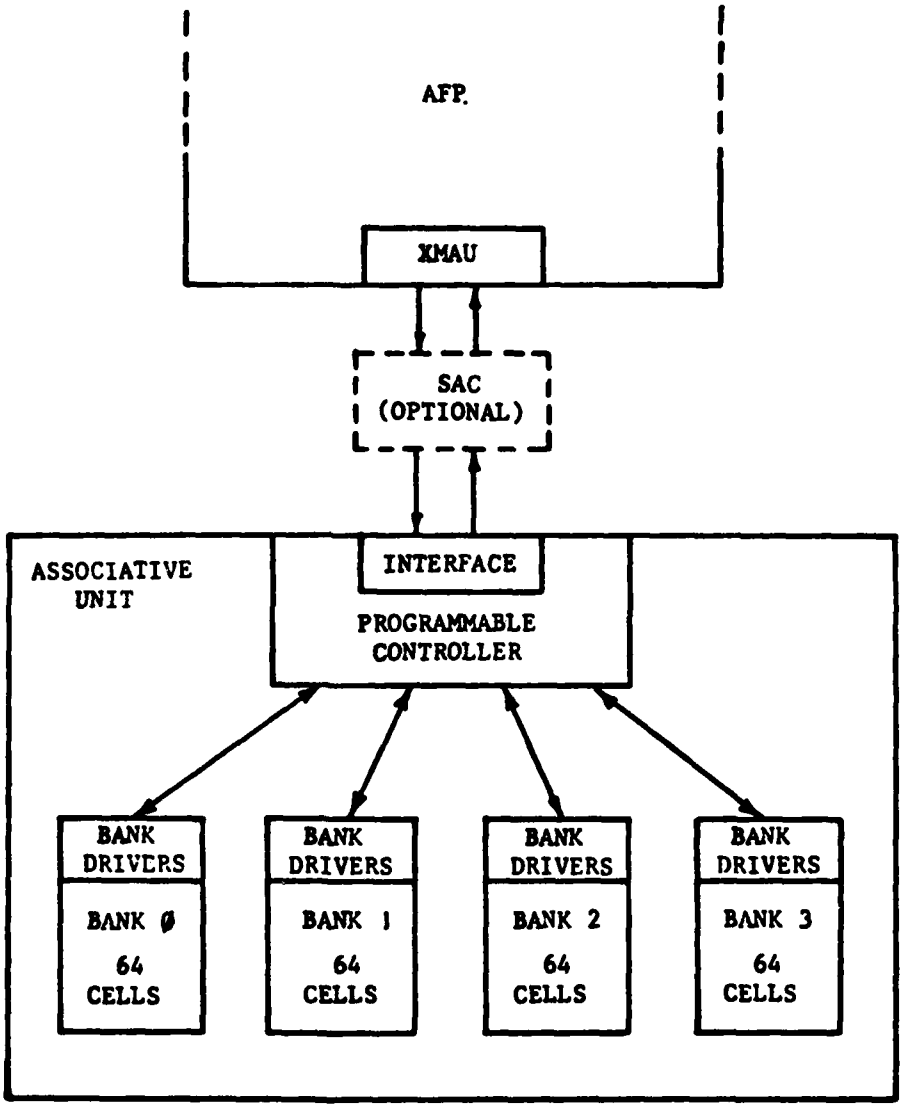


Figure C-1. Organization of the Associative Unit

TABLE C-1. AU/AFP INTERCONNECTIONS

FUNCTION	NUMBER OF BITS
AFP TO AU	
WRITE DATA	64
ADDRESS	3
WRITE	1
REQUEST	1
CLOCK	1
	72
AU to AFP	
READ DATA	20
DATA READY	1
ACKNOWLEDGE	2
	23

C-2.0 ASSOCIATIVE PROCESSING CELLS

The Associative Unit contains 256 associative processing cells designated cell 0 through cell 255. A lower numbered cell is considered to be above a higher numbered cell, and the lowest numbered cell of a collection of cells is considered to be the first cell in that collection.

All cells in an Associative Unit are common to four buses which carry data and controls to the cells and data from any one of them. The names and functions of these buses are given in Table C-2. In addition to the common bus connections, each cell has a set of unique connections for propagating data between adjacent cells, and for future connection of external devices. Each cell also has connections to the response network (see Section C-3).

Each cell in an Associative Unit is in one of two major states: marked or not marked. The response network (see Section C-3) deals only with the marked state of each cell. In addition to the marked state, several other status conditions are defined within each cell and are listed in Table C-3. Various cell operations may be conditioned on the True or False value of these conditions.

The elements of a cell are illustrated in Figure C-2 and include a 256 by 4 bit random access memory (RAM), a 32-function 4-bit arithmetic logic unit (ALU), cell control logic, and various registers and internal buses. The cell elements and their purposes are listed in Table C-4.

The cell operations are controlled by the 20-bit Cell Control Bus (CCB). The various cell operations and their associated control bits are listed in Table C-5 and the cell ALU functions are listed in C-6.

TABLE C-2. COMMON CELL BUSES

NAME	LINES	PURPOSE
CCB	20	CELL CONTROL BUS - These signals define the operation to be performed by the cells. (See Table 2.4 for details.)
CDB	4	CELL DATA BUS - This bus supplies data which may be used as an operand by the cells or be written into the cell memories.
CAB	8	CELL ADDRESS BUS - This bus supplies an address to the cell memories.
CRB	4	CELL RESPONSE BUS - This bus is used to transfer data or results out of the cell array. To avoid bus contention, only the first (lowest numbered) cell in the Marked state is connected to this bus at any given time.

TABLE C-3. CELL STATUS CONDITIONS

NAME	DESCRIPTION
U	UNCONDITIONAL - This condition is always true.
M	MARKED - The cell is in the Marked state.
W	ONES - The ALU output is equal to 1111.*
C	CARRY - The cell Carry register is equal to 1.
P	PROPAGATE - Any cell above the cell in question is in the Marked state.
F	FIRST - The cell in question is not the first (lowest numbered) cell in the Marked state.
AM	ABOVE MARKED - The cell immediately above (lower numbered) the cell in question is in the Marked state.
BM	BELOW MARKED - The cell immediately below (higher numbered) the cell in question is in the Marked state.

This condition is true when ALU inputs A and B are equal and ALU function A minus B minus one is selected (CCB 4, 3, 2, 1, 0 = 00220).

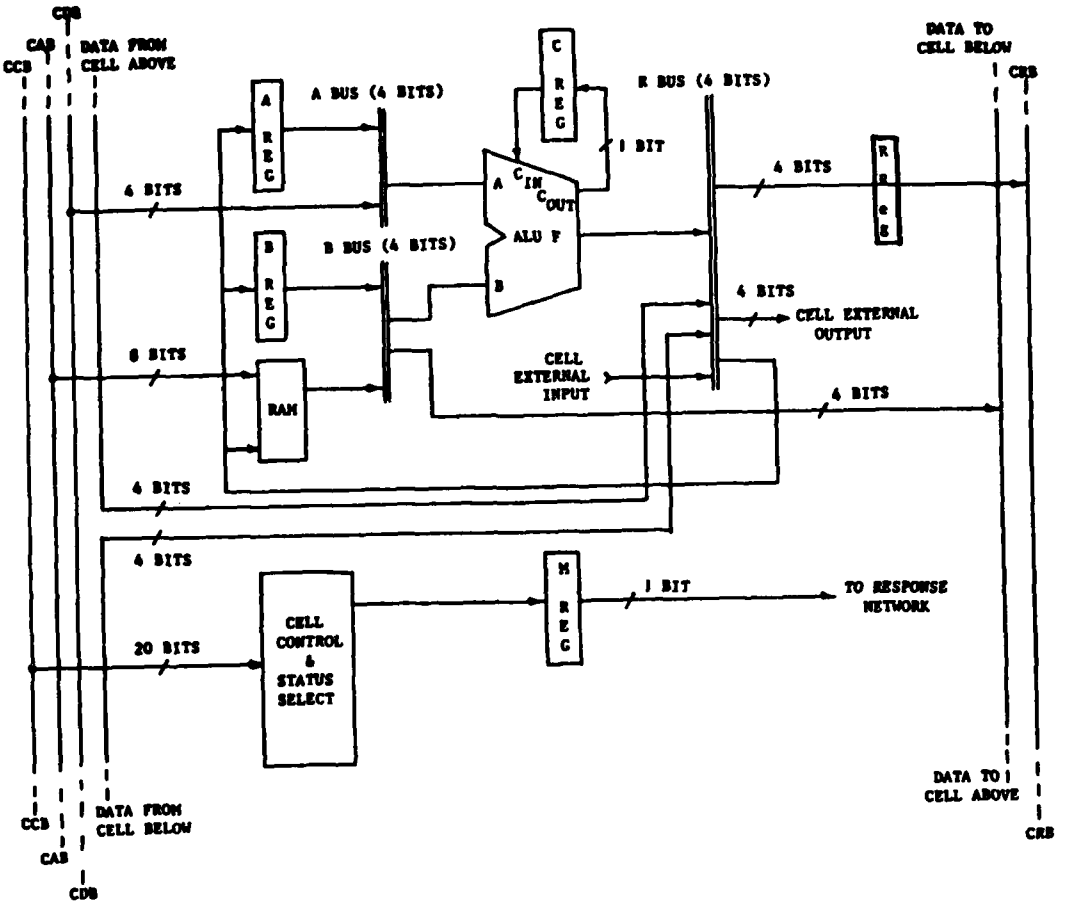


Figure C-2. Associative Processing Cell Structure

TABLE C-4. CELL ELEMENTS

ELEMENT	PURPOSE
A Register	Provides temporary storage for an operand to be placed on the A bus (4 bits).
B Register	Provides temporary storage for an operand to be placed on the B bus (4 bits).
M Register	Contains the Mark status of the cell (1 bit).
C Register	Contains the value of the carry input to the ALU (1 bit).
R Register	Contains a cell operation result to be placed on the R bus.
A Bus	Transmits an operand from either the A Register or the CDB bus to the A input of the ALU (4 bits).
B Bus	Transmits an operand from either the B Register or the RAM data output to the B input of the ALU. Also transmits data to the cells immediately above and immediately below (4 bits).

TABLE C-4. CELL ELEMENTS (Cont'd)

ELEMENT	PURPOSE
R Bus	Transmits data from the ALU output or the cell immediately above or the cell immediately below or external terminals to the A Register, B Register, RAM, data input, CRB bus, and external terminals (4 bits).
RAM	256 x 4 bit Random Access Memory provides storage for a list of operands.
ALU	32 function Arithmetic Logic Unit (4 bits) performs logical and arithmetic functions on two operands.
Control	Decodes the 20-bit CCB bus signals and determines the operation of other cell elements based on cell status conditions.

TABLE C-5. CELL OPERATIONS

CCB BITS	CELL ELEMENT	FUNCTION CODE	FUNCTIONS
19	Status Select	--	Use complement of cell status when set.
18,17,16	Status Select	000	U
		001	M
		010	W
		011	C (see Table C-3)
		100	$\overline{P}$
		101	$\overline{F}$ **
		110	AM
		111	BM
15,14	Carry Register*	00	NOP
		01	CLEAR
		10	SET
		11	LOAD from ALU carry output
13,12	Mark Register*	00	NOP
		01	SET
		10	CLEAR
		11	COMPLEMENT
11	B Register	--	Load B Register from R Bus when set
10	A Register	--	Load A Register from R Bus when set

* Operation conditional on selected cell status condition.

** Due to propagation delays, this condition may not be valid until the second cycle following a change in state of a mark flip-flop.

TABLE C-5 CELL OPERATIONS (Cont'd)

CCB BITS	CELL ELEMENT	FUNCTION CODE	FUNCTIONS
9	B Bus	0	B Bus source is RAM output data
		1	B Bus source is B Register
8	A Bus	0	A Bus source is CDB
		1	B Bus source is A Register
7,6	R Bus, R Reg.	00	R Bus source is ALU, R Reg not clocked
		01	R Bus source is ALU**
		10	R Bus Source is lower cell B Bus**
		11	R Bus Source is upper cell B Bus**
			** R-Reg clocked
5	RAM*	--	Write RAM from R Bus when set
4	ALU Mode	0	ALU in Logic Mode (see Table
		1	ALU in Arithmetic mode (see Table C-6)
3,2,1,0	ALU Function	--	See Table C-6

* Operation conditional on selected cell status condition.

** R-Reg is loaded, but B-Reg is enabled onto CRB only if cell is the first marked.

TABLE C-6. CELL ALU FUNCTIONS

	CCB 4 = 1	CCB 4 = 0; ARITHMETIC OPERATIONS	
CCB BITS 3 2 1 0	LOGIC FUNCTIONS	C = 0 (no carry)	C = 1 (with carry)
0 0 0 0	$F = \overline{A}$	F = A	F = A PLUS 1
0 0 0 1	$F = \overline{A + B}$	F = A + B	F = (A + B) PLUS 1
0 0 1 0	$F = \overline{A}B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
0 0 1 1	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
0 1 0 0	$F = \overline{AB}$	$F = A$ PLUS $A\overline{B}$	$F = A$ PLUS $A\overline{B}$ PLUS 1
0 1 0 1	$F = \overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$	$= (A + B)$ PLUS $A\overline{B}$ PLUS 1
0 1 1 0	F = A + B	F = A MINUS B MINUS 1*	F = A MINUS B
0 1 1 1	$F = A\overline{B}$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
1 0 0 0	$F = \overline{A} + B$	F = A PLUS AB	F = A PLUS AB PLUS 1
1 0 0 1	$F = \overline{A + B}$	F = A PLUS B	F = A PLUS B PLUS 1
1 0 1 0	F = B	$F = (A + \overline{B})$ PLUS AB	$F = (A + \overline{B})$ PLUS AB PLUS 1
1 0 1 1	F = AB	F = AB MINUS 1	F = AB
1 1 0 0	F = 1	F = A PLUS A**	F = A PLUS A PLUS 1
1 1 0 1	$F = A + \overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
1 1 1 0	F = A + B	$F = (A + \overline{B})$ PLUS A	$F = (A + \overline{B})$ PLUS A PLUS 1
1 1 1 1	F = A	F = A MINUS 1	F = A

* When this function is selected, cell condition A=B is true if ALU inputs A and B are equal.

** This function shifts each bit to the next more significant position.

C-3.0 THE RESPONSE NETWORK

The Response Network is hierarchically organized, using a few types of logic modules. At the lowest level, each cell has an associated Rail Module. The next level is comprised of Block Response Modules, and above that, Bank Response Modules. In addition to providing an indication to the controller, R, when any cell of the system is marked, the Response Network also implements the rails used for cell-to-cell intercommunication. It is the topology of the Response Network that is responsible for the organization of the Associative Processing Cells into blocks and banks.

C-3.1 Block Response Modules

A Block Response Module serves a block of eight cells and contains eight Rail Modules and one Response Collector Module with the interconnection as shown in Figure C-3. The terminals at the top and bottom of the Block Response Module are for continuation of the rails. The output BR (Block Response), which is produced by the Response Collector Module (Figure C-4), indicates whether any cell of the block is marked. The Response Collector Module is also used to provide a high-speed route for the propagate rail.

C-3.2 Rail Modules

The internal construction of a Rail Module is shown in Figure C-5. As in the case of the block levels, the terminals on the top and bottom support the intercommunication rails. Each Rail Module has one input from the cell, the contents of its Mark Register, M, and four outputs to its associated cell. These outputs are the status conditions propagate P; Above Marked, AM; Below Marked, BM; and First Marked, F.

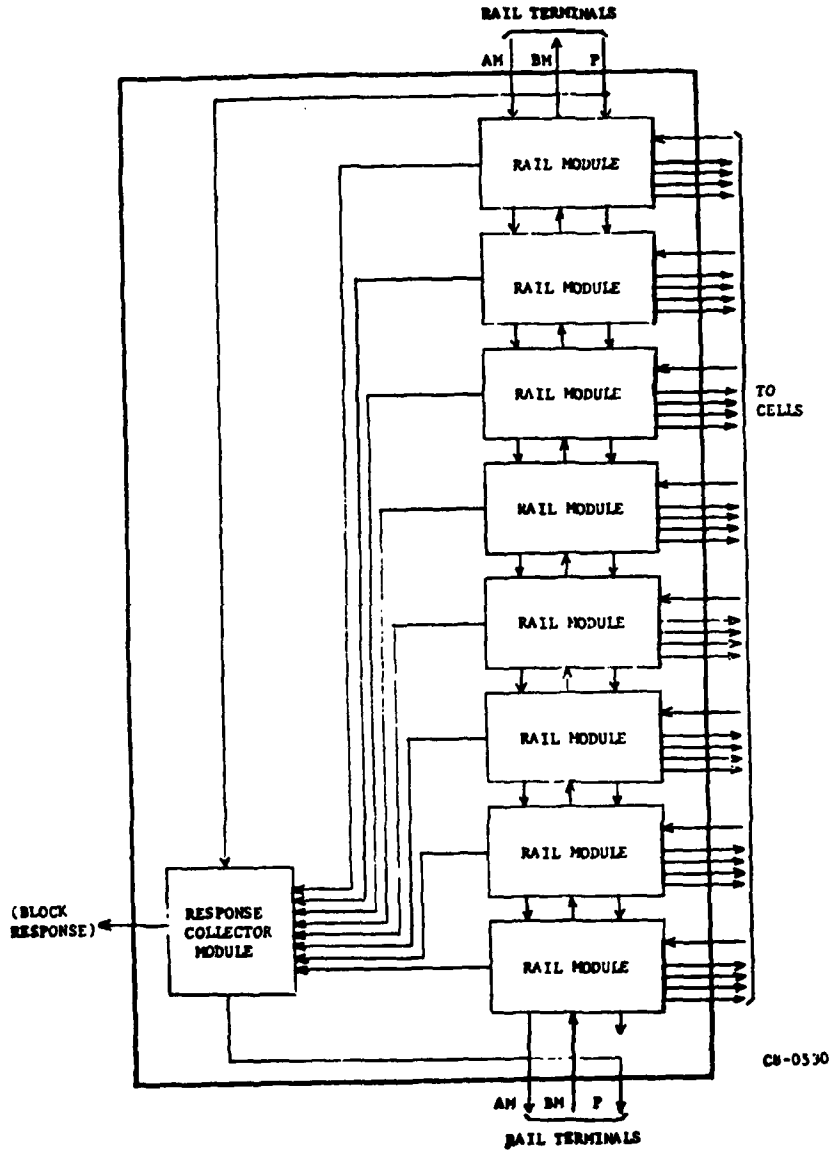


Figure C-3. Block Response Module

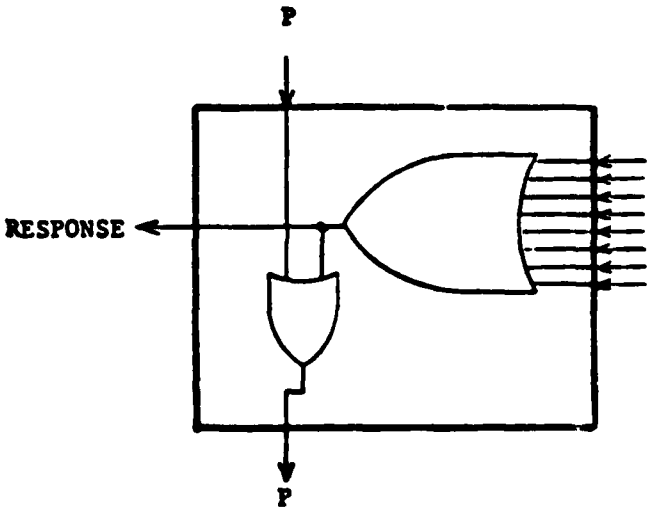


Figure C-4 Response Collector Module

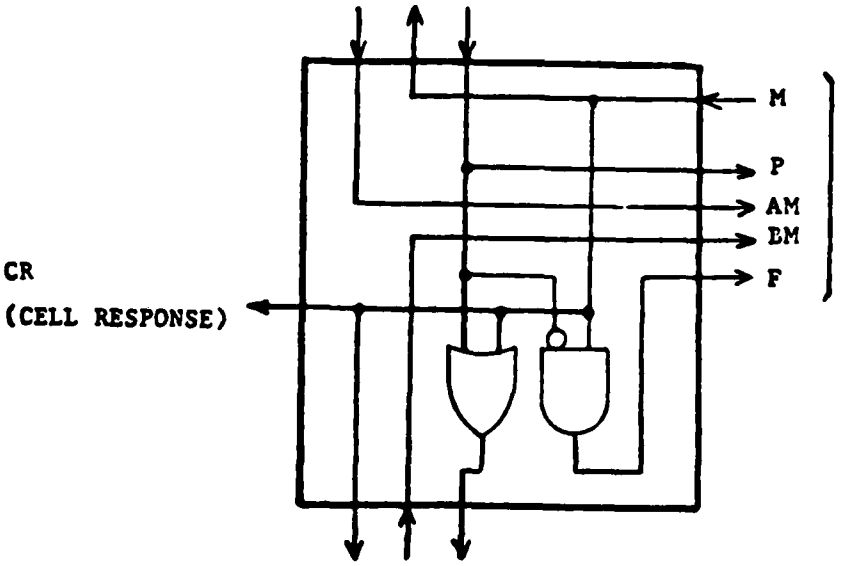


Figure C-5. Rail Module

C-3.3 Bank Response Modules

The Bank Response Module is merely the interconnection of the eight Block Response Modules associated with the blocks of a bank of cells, together with one Response Collector Module. The wiring of a Bank Response Module is shown in Figure C-6.

C-3.4 Memory Response Module

An Associative Unit contains one Memory Response Module consisting of four Bank Response Modules and other circuitry as shown in Figure C-7. Signal IP from the controller defines the initial condition of the Propagate, P, and Above Marked, AM, Rails.

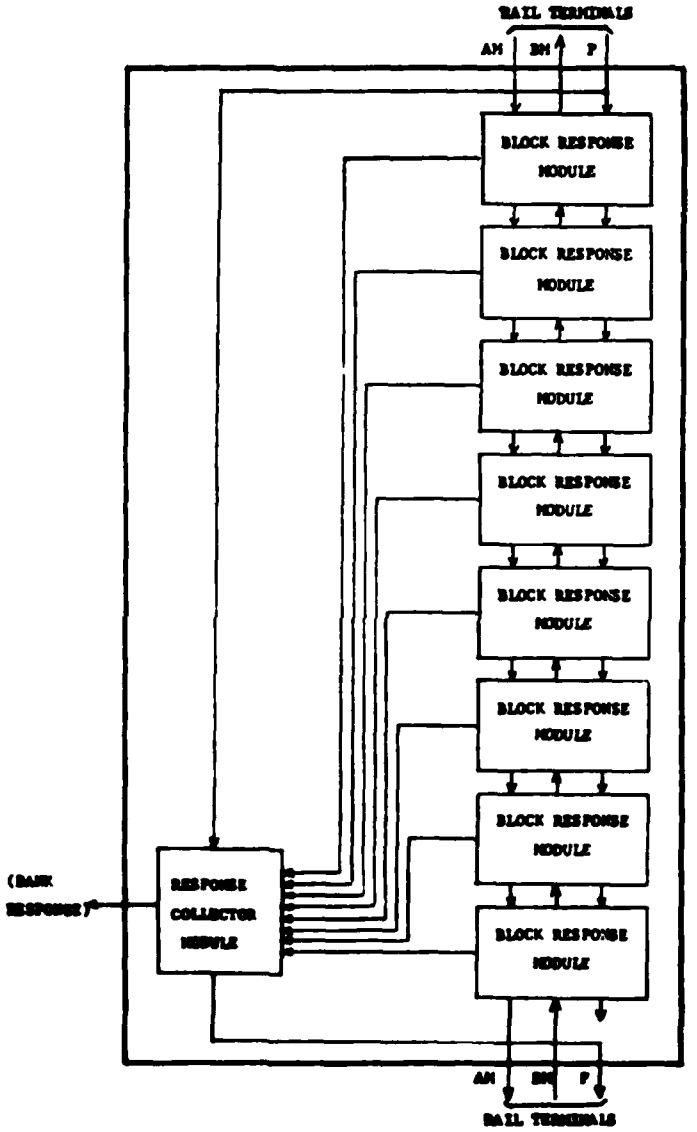


Figure C-6. Bank Response Module

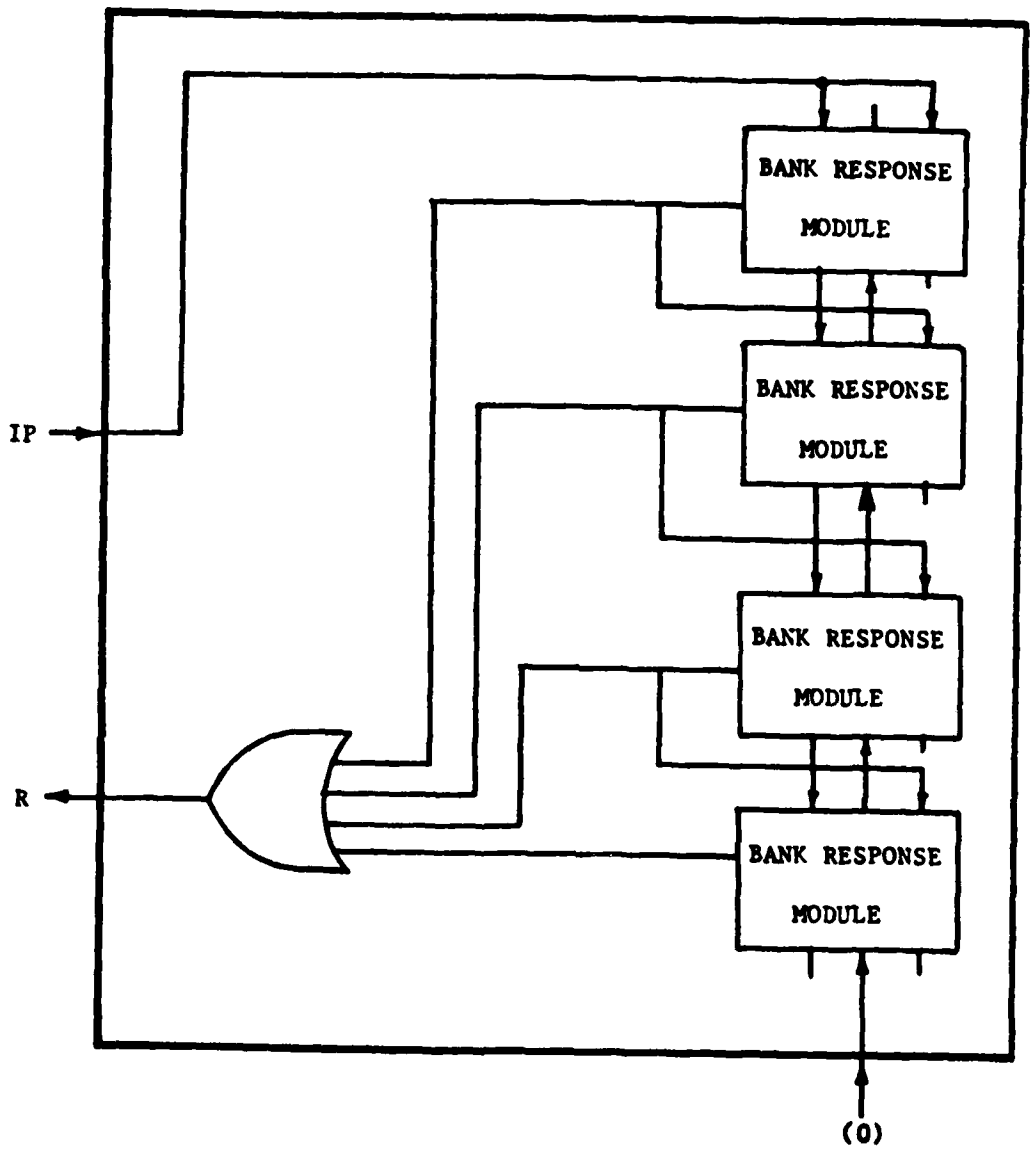


Figure C-7. Memory Response Module

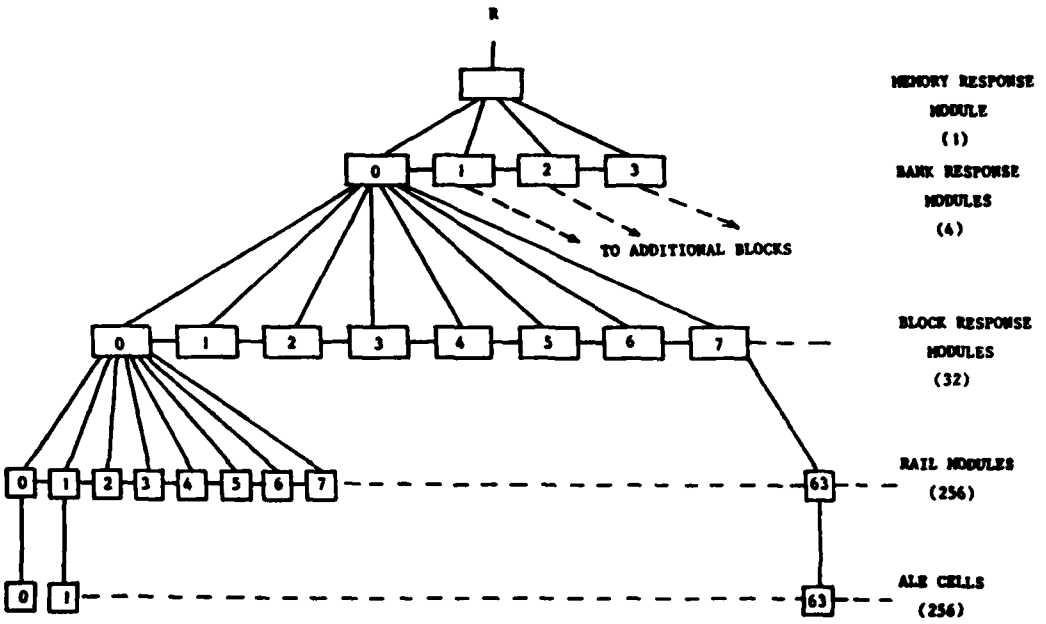


Figure C-8. Response Network Hierarchy

C-4.0 CONTROLLER

The Associative Processing Cells are controlled by a microprogrammed controller. The controller is connected to the AFP either directly to the XMAU or indirectly though the SAC. By initiating a read or write to the Associative Unit the AFP can load or read back the AU microprogram memory, examine AU registers, select the AU operating mode, start a micro-program; or transfer data to and from the AU.

The controller consists of a 256-word by 48-bit microprogram memory, several registers, and associated interface and control logic. A block diagram showing the functional organization of the controller appears as Figure C-9. Table C-7 shows the controller registers and their functions.

C-4.1 Interface Connections

The AFP initiates all communications with the AU with either a write or a read request on the HPR port. The action and response by the AU is determined by the address and/or data associated with the request. Only the three least significant bits of the address are decoded by the AU. Up to 64 bits of information may be transmitted to the AU on a write operation. The 4-bit contents of the AU control/status register (CTL) and up to 16 bits of data are returned to the AFP on a read operation.

Each request by the AFP to the AU must be acknowledged by the AU before another request is made. Violations of this sequence may cause improper operation. The AU will not acknowledge any request until the requested operation is performed. Acknowledges will be issued by the AU as described in Table C-8. The acknowledge consists of a sequence of signals to the AFP as defined in Section C-5.

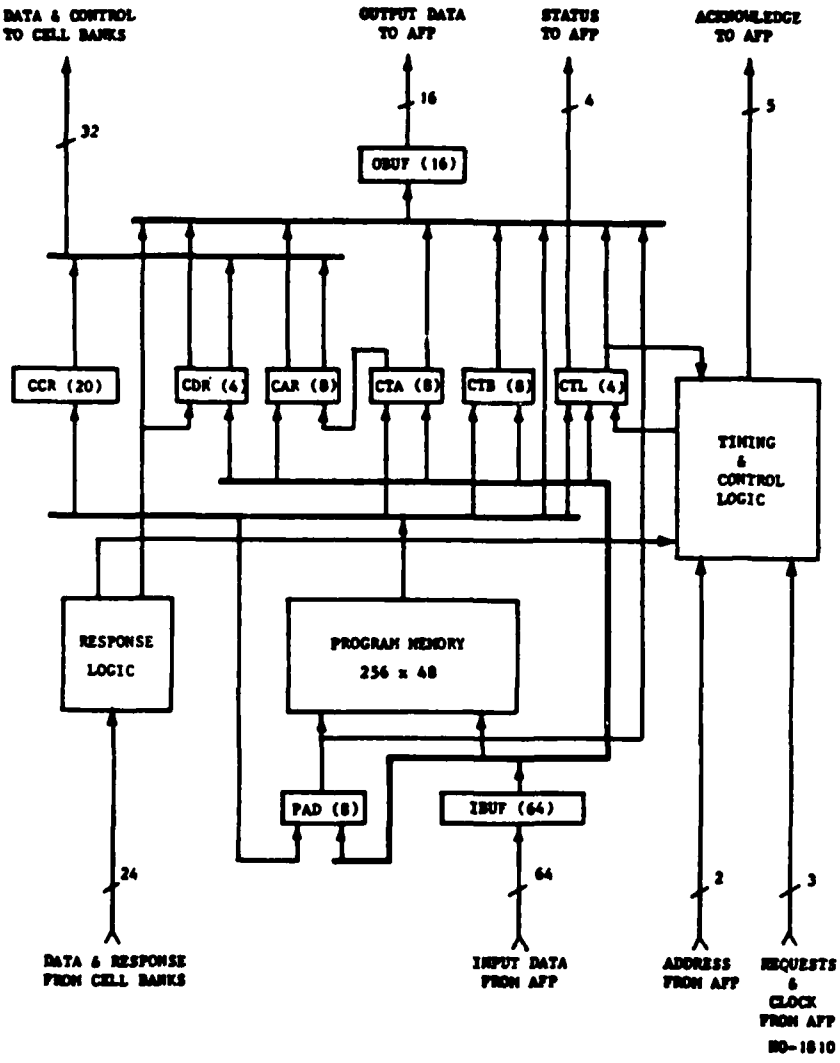


Figure C-9. AU Controller Block Diagram

TABLE C-7. CONTROLLER REGISTERS

NAME	LENGTH	USE
IBUF	64	INPUT BUFFER REGISTER
OBUF	16	OUTPUT BUFFER REGISTER
PAD	8	MICROPROGRAM ADDRESS REGISTER
CTA	8	COUNTER A, USED BY PROGRAMS
CTB	8	COUNTER B, USED BY PROGRAMS
CAR	8	CELL ADDRESS REGISTER, BROADCAST TO ALL CELLS
CDR	4	CELL DATA REGISTER, BROADCAST TO ALL CELLS
CCR	20	CELL CONTROL REGISTER, BROADCAST TO ALL CELLS
CTL	4	CONTROL INPUT FROM AFP & AU STATUS TO AFP

TABLE C-8. AU/AFP INTERFACE OPERATIONS

ADDRESS	REQUEST	AU STATUS	ACTION BY AU
0	READ	$\overline{\text{RUN}}$, $\overline{\text{OUTPUT RDY}}$	AFP Read Data(44..47) <-- CTL(3..0); ACKNOWLEDGE
0	READ	$\overline{\text{RUN}}$, OUTPUT RDY	AFP Read Data(44..47) <-- CTL(3..0); AFP Read Data(48..63) <-- OBUF(15..0); Clear OUTPUT RDY; ACKNOWLEDGE
0	READ	RUN, $\overline{\text{OUTPUT RDY}}$	Wait for OUTPUT RDY or $\overline{\text{RUN}}$, then: AFP Read Data(44..47) <-- CTL(3..0); AFP Read Data(48..63) <-- OBUF(15..0); Clear OUTPUT RDY; ACKNOWLEDGE
0	READ	RUN, OUTPUT RDY	AFP Read Data(44..47) <-- CTL(3..0); AFP Read Data(48..63) <-- OBUF(15..0); Clear OUTPUT RDY; ACKNOWLEDGE
0	WRITE	$\overline{\text{RUN}}$	IBUF(63..0) <-- AFP Write Data(0..63); Clear INPUT READY; ACKNOWLEDGE
0	WRITE	RUN, $\overline{\text{INPUT RDY}}$	ACKNOWLEDGE
0	WRITE	RUN, INPUT RDY	IBUF(63..0) <-- Write Data(0..63); Clear INPUT READY: Wait for INPUT RDY or $\overline{\text{RUN}}$, then ACKNOWLEDGE
1	READ	$\overline{\text{RUN}}$	AFP Read Data(44..47) <-- CTL(3..0); AFP Read Data(48..55) <-- PAD(7..0); AFP Read Data(60..63) <-- CTL(3..0); ACKNOWLEDGE
1	READ	RUN	AFP Read Data(44..47) <-- CTL(3..0); ACKNOWLEDGE
1	WRITE	$\overline{\text{RUN}}$	PAD(7..0) <-- AFP Write Data(48..55); CTL(1..0) <-- AFP Write Data(62..63); ACKNOWLEDGE
1	WRITE	RUN	CTL(1..0) <-- AFP Write Data(62..63); ACKNOWLEDGE

TABLE C-8. AU/AFP INTERFACE OPERATIONS (Cont'd)

ADDRESS	REQUEST	AU STATUS	ACTION BY AU
2	READ	$\overline{\text{RUN}}$	AFP Read Data(44..47) <-- CTL(3..0); AFP Read Data(48..55) <-- CTB(7..0); AFP Read Data(56..63) <-- CTA(7..0); ACKNOWLEDGE
2	READ	$\overline{\text{RUN}}$	AFP Read Data(44..47) <-- CTL(3..0); ACKNOWLEDGE
2	WRITE	-	ACKNOWLEDGE
3	READ	$\overline{\text{RUN}}$	AFP Read Data(44..47) <-- CTL(3..0); AFP Read Data(50..63) <-- CAR(7..0); ACKNOWLEDGE
3	READ	RUN	AFP Read Data(44..47) <-- CTL(3..0); ACKNOWLEDGE
3	WRITE	-	ACKNOWLEDGE
ADDRESS	REQUEST	AU STATUS	ACTION BY AU
4	READ	$\overline{\text{RUN}}$	AFP Read Data (44..47)<-- XTL (3..0); AFP Read Data (48..63)<-- PGM (PAD)*;
4	READ	RUN	ACKNOWLEDGE
4	WRITE	$\overline{\text{RUN}}$	PGM (PAD)**<-- AFP WRITE Data (48..63); ACKNOWLEDGE
4	WRITE	RUN	ACKNOWLEDGE

* See Table C-9
** See Table C-10

All data transferred from the AFP to the AU are loaded into the controller's 64 bit Input Buffer (IBUF). This buffer may be right shifted and loaded into the cell data register four bits (one nibble) per cycle by the AU microprogram. THe AU microprogram may cause Counter A, Counter B or the Cell Address Register to be loaded from the least significant bits of IBUF, but IBUF should not be shifted before these operations. THE PAD register, the CTL register, and the Program Memory may be forceloaded from the least significant bits of IBUF by the AFP.

All data transferred from the AU to the AFP are loaded into the 16-bit Output Buffer (OBUF) for transmission. This buffer may be left shifted and loaded four bits (one nibble) per cycle by the AU microprogram. The contents of various controller registers or of locations in the Program Memory may be loaded into OBUF under AFP control.

The contents of the controller four-bit Control/Status register (CTL) are transmitted to the AFP along with the contents of OBUF on each Read operation. The CTL register bits are defined in Figure C-10.

Allowable interface operations are defined in Table C-10. Section C-5 contains details on the interface signals.

C-4.2 Functional Modes

The controller has two functional modes: Wait and Run. Selection of either one of these modes is determined by the state of bit zero of the control register (CTL).

In the Wait Mode the AU is in a halt state. The AU will output the contents of the CAR, CTA, CTB, PAD, or CTL registers on request by the AFP. The program memory can be loaded or read by the AFP in the wait

TABLE C-9. REPROGRAM SEQUENCE

Initial Contents of PAD = P

CYCLE	OPERATION
0	PGM(P;47..32)*<-- AFP Write Data(56..63)
1	PGM(P;31..16) <-- AFP Write Data(56..63)
2	PGM(P;15..0) <-- AFP Write Data(56..63)
3	PGM(P+1;47..32) <-- AFP Write Data(56..63)
4	PGM(P+1;31..16) <-- AFP Write Data(56..63)
.	.
.	etc.
.	.

TABLE C-10. INTERROGATE SEQUENCE

Initial Contents of PAD = P

CYCLE	OPERATION
0	AFP Read Data(56..63) <-- PGM(P;47..32)*
1	AFP Read Data(56..63) <-- PGM(P;31..16)
2	AFP Read Data(56..63) <-- PGM(P;15..0)
3	AFP Read Data(56..63) <-- PGM(P;47..32)
4	AFP Read Data(56..63) <-- PGM(P;31..16)
.	.
.	etc.
.	.

*NOTE: Pad must be loaded to reset segment counter to zero.

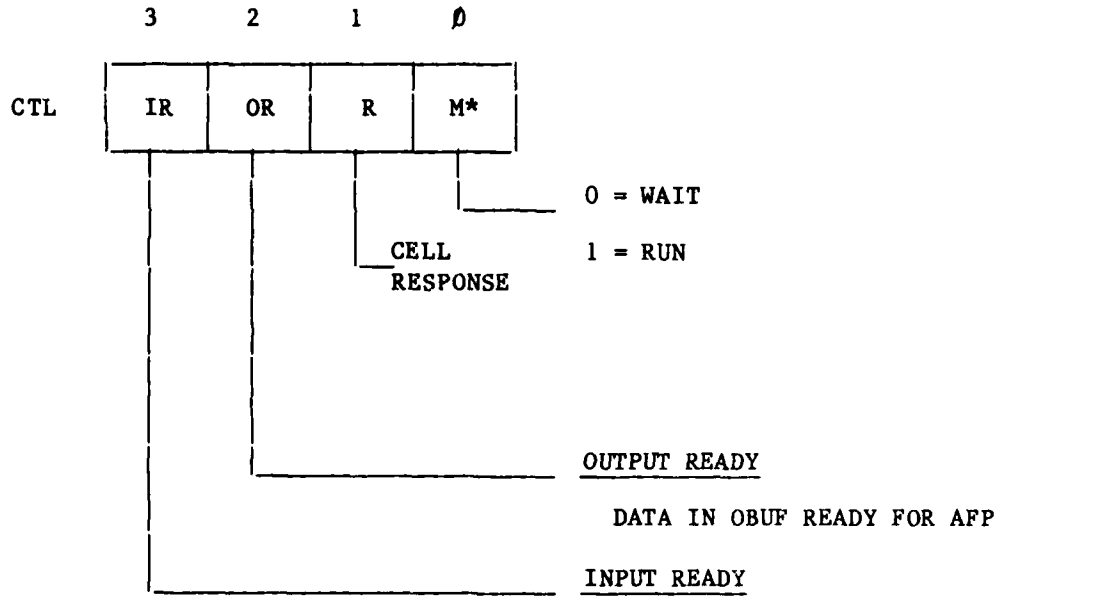


Figure C-10. AU Control/Status Register

mode. Table C-8 defines the allowed operations on the AFP/AU interface. Tables C-9 and C-10 define the sequence of loading and reading the AU Program Memory by the AFP.

In Run Mode the AU is under control of a program in the Program Memory. Execution of the program begins with the instruction located at the address contained in the PAD register when Operate Mode is selected. The AU will remain in Operate Mode until the contents of CTL are changed by the AFP or the Wait Mode is entered by program command. Table C-8 defines the allowed operations on the AFP/AU interface.

C-4.3 Controller Microinstructions

In the Run Mode the AU is under the control of a microprogram residing in the Program Memory (PGM). The instruction fields of the 48-bit control words are shown in Table C-11. The various operations that can be executed from one instruction word are listed in Table C-12 along with their codes.

All commands to the controller are executed at the end of the 120 nsec. AU cycle during which the commands were read from PGM. All commands to the Associative Processing Cells (microinstruction bits 28-47) are executed the following cycle. Testing of the Response Network or loading of data from the Cell Response Bus by the controller must not be done until the second cycle following the instruction execution by the controller.

TABLE C-11. AU MICROINSTRUCTION FIELDS

BIT	FUNCTION
7 - 0	BRANCH ADDRESS/IMMEDIATE DATA
11 - 8	BRANCH CONDITION
12	PROGRAM ADDRESS CONTROL
13	RUN CONTROL
15 - 14	OUTPUT BUFFER CONTROL
17 - 16	COUNTER A CONTROL
19 - 18	COUNTER B CONTROL
21 - 20	INPUT BUFFER CONTROL
23 - 22	CELL DATA CONTROL
26 - 24	CELL ADDRESS CONTROL
27	INITIAL PROPAGATE VALUE
47 - 28	CELL CONTROL WORD

TABLE C-12. AU MICROINSTRUCTION CODES

BRANCH CONDITIONS - BITS 8 - 11

10	9	8	
0	0	0	LOGIC 1
0	0	1	CELL RESPONSE
0	1	0	INPUT READY (INPUT BUFFER EMPTY)
0	1	1	OUTPUT READY (OUTPUT BUFFER FULL)
1	0	0	COUNTER A=0
1	0	1	COUNTER B=0
1	1	0	COUNTER A=COUNTER B
1	1	1	CAR = 0

11	
0	USE TRUE VALUE OF SELECTED BRANCH CONDITION
1	USE FALSE VALUE OF SELECTED BRANCH CONDITION

PROGRAM ADDRESS CONTROL - BIT 12

12	
0	IF BRANCH CONDITION TRUE EXECUTE NEXT INSTRUCTION, ELSE REPEAT CURRENT INSTRUCTION
1	IF BRANCH CONDITION TRUE JUMP TO ADDRESS CONTAINED IN BITS 0-7, ELSE EXECUTE NEXT INSTRUCTION

TABLE C-12. AU MICROINSTRUCTION CODES (Cont'd)

RUN CONTROL - BIT 13

13	
0	NO OPERATION
1	SET WAIT MODE

OUTPUT BUFFER CONTROL - BITS 14-15

14	
0	NO OPERATION
1	SHIFT LEFT 4 BITS AND LOAD LOWER 4 BITS FROM CELL READ BUS (CRB)

15	
0	NO OPERATION
1	SET OUTPUT READY

COUNTER A CONTROL - BITS 16-17

17	16	
0	0	NO OPERATION
0	1	LOAD FROM BITS 0-7 OF PROGRAM MEM w FIELD
1	0	LOAD FROM BITS 0-7 OF INPUT BUFFER*
1	1	DECREMENT

* Input Buffer must not be shifted between loading IBUF and executing this instruction.

TABLE C-12. AU MICROINSTRUCTION CODES (Cont'd)

COUNTER B CONTROL - BITS 18-19

19	18	
0	0	NO OPERATION
0	1	LOAD FROM BITS 0-7 OF PROGRAM MEM u FIELD
1	0	LOAD FROM BITS 0-7 OF INPUT BUFFER*
1	1	DECREMENT

21	
0	NO OPERATION
1	SET INPUT BUFFER READY (EMPTY) F-F

20	
0	NO OPERATION
1	RIGHT SHIFT ONE NIBBLE (4 BITA)

CELL DATA BUS CONTROL - BITS 22-23

23	22	
0	0	NO OPERATION
0	1	LOAD FROM BITS 0-3 OF INPUT BUFFER
1	0	LOAD FROM CELL RESPONSE BUS CRB
1	1	LOAD FROM u FIELD CAN GET CLEAR FROM THE ALU F=0

* Input Buffer must not be shifted between loading IBUF and executing this instruction.

TABLE C-12. AU MICROINSTRUCTION CODES (Cont'd)

CELL ADDRESS CONTROL - BITS 24-26

26 25 24	
000	NO OPERATION
001	LOAD FROM BITS 0-7 OF INPUT BUFFER*
010	LOAD FROM COUNTER A
011	CLEAR
100	INCREMENT
101	DECREMENT
110	NO OPERATION
111	NO OPERATION

INITIAL PROPAGATE VALUE - BIT 27

The value of this bit is placed on the initial propagate input of the response network.

CELL CONTROL WORD 0-19 - BITS 28-47

See Table C-5

* Input Buffer must not be shifted between loading IBUF and executing this instruction.

C-5.0 INTERFACE SIGNALS

The Associative Unit is connected to the AFP through the High Performance RAM (HPR) interface. Connection to this interface can be either at the SAC unit or the XMAU unit of the AFP (see Figure C-1). All interface signals are single-ended ECL unidirectional. All interface connections are made with 75-ohm coaxial cable. The interface signals are described below.

Signals from AFP to AU

Memory Request (1)
A logical one indicates a request for an Associative Unit cycle (read or write). The signal is a pulse with a duration of one AFP clock period.

Write (1)
Logical one indicates that the current request is for an AU write operation. Zero indicates a read. The signal is valid during the time the Memory Request signal is a one.

Address (2)
These lines specify the address for the requested AU operation. The lines are valid during the Memory Request signal.

Write Data (64)
Sixty-four bits of data to be written at the location specified by the address lines. The data lines become valid one clock period after the request signal, and remain valid until the next AU cycle.

Clock (1)
One line carrying the system master clock signal. This is a free-running square wave, with a period equal to the system clock period (20 nsec.) and with equal one and zero times.

Signals from AU to AFP

Data Ready (1)
A logical one on this line signifies the presence of valid data on the Read Data lines. The signal is a pulse with a duration of one clock period.

Read Data (20)
Twenty lines which transmit data from the AU to the receiving device on memory read operations. They are valid during the data ready pulse.

Request Acknowledge (2)
A clock period pulse which is transmitted when the AU has processed a current operation to the point at which it is ready to receive another request. This signal is sent one clock period before the Data Ready signal. The signal is returned on both read and write operations. Two lines are provided with this signal.

APPENDIX D

SYSTEM SOFTWARE DESCRIPTION

TABLE OF CONTENTS

L I S T O F F I G U R E S

L I S T O F T A B L E S

D-1.0 SCOPE

This specification defines the operation of the controlware in the "Disk Storage Subsystem" for the Retriever Project demonstration.

D-2.0 APPLICABLE DOCUMENTS

Control Data Spec 64065800 Control Data 854.
Disk Storage Drive Product Specification

Control Data 84000003B Compass 3 Reference Manual

Control Data Spec 77978008A 7000 Channel
Ring Port Engineering Specification

Demonstration Document Retriever System
System Software Description

Control Data Spec xxxxxxxx Advance Flexible Processor

Control Data Spec 77900504 Advance Flexible Processor
System Software Description

Control Data Spec 77900507 Advance Flexible Processor
Application Programmer Reference Manual

D-3.0 CONFIGURATION

The Disk Storage Subsystem (DSS) is connected to the interface ring on the AFP Array Configuration System. In addition to the DSS, the interface ring also has the Interface Advance Flexible Processor (IAFP) and the PDP-11/70 host processor connected to it.

The DSS has a 7000 channel Ring Port which is connected to the interface ring. The 7000 Channel Ring Port is also connected to one of the peripheral processor units (PPU's). It receives control commands and data from sources connected to the interface ring and disassembles and passes the commands and data to the PPU. It also assembles data received from the PPU and transmits this data over the interface ring.

The DSS controlware is executed by 3 PPU's and is stored in their internal memories.

The PPU connected to the 7000 Channel Ring Port is the controlling PPU. It interprets the control commands from the interface ring, writes data from the interface ring into LCM, reads LCM data, transmits data and status information to the interface ring, and directs disk I/O activities.

There are two other PPU's that are utilized in the DSS. Each is connected to an 857 disk drive and controls the disk. At the direction of the controlling PPU, they read or write disks at specified track positions and they obtain status information on the disks. The following summarize the characteristics of the 857 disks:

- 8 bits per byte
- 192 bytes per sector
- 16 sectors per track
- 10 tracks per cylinder
- 200 cylinders per disk

LCM provides for 480K bytes* of storage that is used for intercommunications between the PPU's and to buffer data for disk I/O activity and as storage for the application. Figure D-3.1 shows the interface ring configuration along with the disk storage subsystem.

* LCM word sizes are actually 12 bits per word and LCM has 320K words. This means that the LCM address space is 320K addresses.

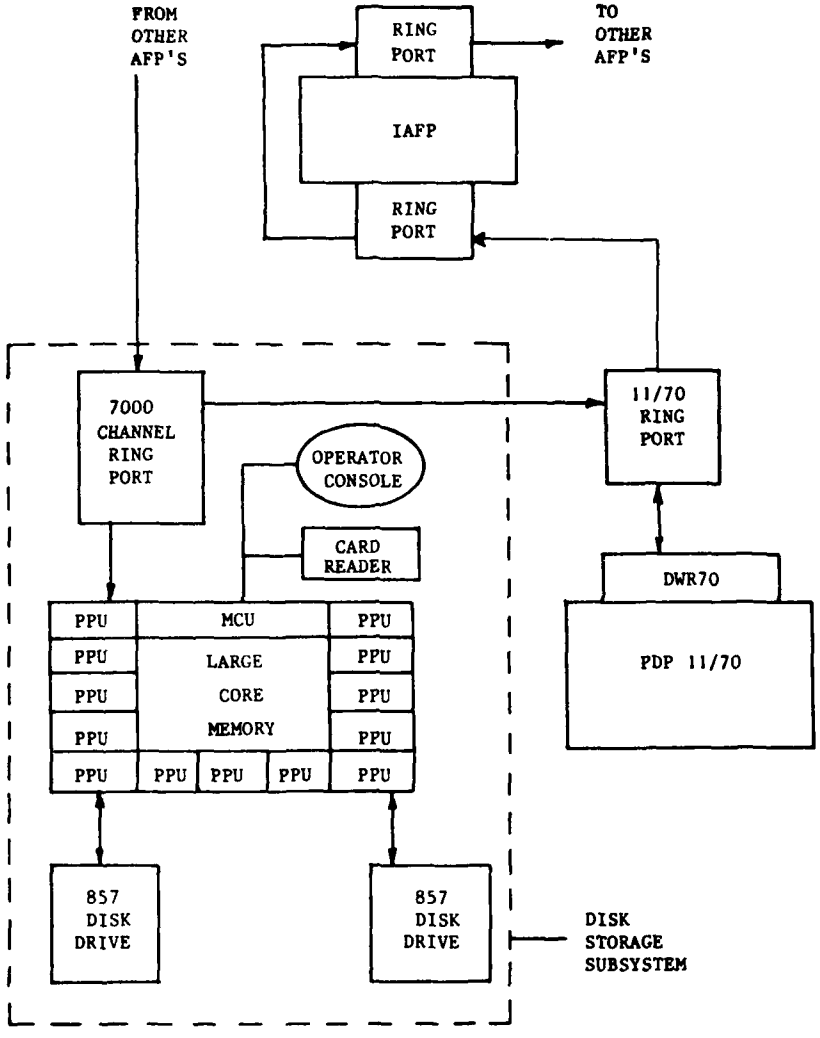


Figure D-3-1. Interface Ring Configuration

D-4.0 FACILITIES

To support the IAFP's and PDP's ability to read and write LCM and to drive the disks in an efficient manner, 3 facilities are required. They are a means of addressing LCM, of assembling and disassembling words for LCM and of statusing the operation of the disks.

D-4.1 ADDRESSING LCM

The mode of addressing LCM is indirect. A group of 40 pointers is provided for the indirect addressing of LCM. These pointers can be set by the IAFP and PDP and will automatically be incremented after the location which they are pointing to has been referenced.

D-4.2 ASSEMBLY/DISASSEMBLY

Since the IAFP and the PDP accepts and transmits 16-bit data words to the DSS while LCM accepts and transmits 12-bit data words, some means of assembly and disassembly is required. The 7000 channel ring port does provide 4 modes of assembly and disassembly and in terms of disk storage utilization the most efficient of these modes is to have 3 16-bit data words from the IAFP or the PDP disassembled into 4 12-bit data words for LCM and 4 12-bit data words from LCM assembled into 3 16-bit data words for transmission over the interface ring. Figure D-4-1 shows how 3 16-bit words from the IAFP or PDP would be disassembled into 4 12-bit words. Figure D-4-2 shows how 4 12-bit words from LCM would be assembled into 3 16-bit words.

D-4.3 STATUS WORDS

A 16-bit status word is available to IAFP and the PDP for monitoring disk drive activity. It is transmitted along with the two disk track code words to the interface ring. Figure D-4-3 shows the format of the 3 16-bit status words transmitted to the interface ring.

```
                                                                    IAFP
AAAAAAAAAAAAAAAA  BBBBBBBBBBBBBBBB  CCCCCCCCCCCCCCCC               or
                                                                    PDP

AAAAAAAAAAAA  AAAABBBBBBBB  BBBBBBBBCCCC  CCCCCCCCCCCC             LCM
```

Figure D-4-1. The Disassembly of IAFP or PDP words

```
AAAAAAAAAAAA  BBBBBBBBBBBB  CCCCCCCCCCCC DDDDDDDDDDDD             LCM

                                                                    IAFP
AAAAAAAAAAAABBBB  BBBBBBBBCCCCCCCC  CCCCDDDDDDDDDDDD              or
                                                                    PDP
```

Figure D-4-2. The Assembly of LCM Words

STATUS DISK1	STATUS DISK0	ERROR CODE
XXXX	CODE WORD	DISK1
XXXX	CODE WORD	DISK0

Figure D-4-3. Status Words to Ring Format

The status word indicates whether or not the drives are busy, found the selected cylinder and encountered problems during data transfer operations. The code words identify the last disk track processed. In addition to being able to examine the status words, the IAFP and PDP can clear or set selected bits. Figure D-4-4 shows the organization of the status word.

D-4.3.1 DRIVE BUSY

When this bit is set to 1, it indicates that the disk drive is busy either reading the disk or writing on the disk.

D-4.3.2 SEEK ERROR ENCOUNTERED

When this bit is set to 1, it indicates that there was a seek error encountered in the process of seeking a cylinder.

D-4.3.3 PACK UNSAFE OR NOT READY

When this bit is set to 1, it indicates that the selected disk drive has one or more fault conditions and/or an unavailable condition has occured.

D-4.3.4 CHECKSUM ERROR ENCOUNTERED

When this bit is set to 1, it indicates that during a disk read operation 5 attempts were made to read a sector and to match the computed checksum value to the recorded checksum value and each time the match did not occur.

D-4.3.5 HEADER ERROR ENCOUNTERED

When this bit is set to 1, it indicates that during a read or write operation the desired header did not match.

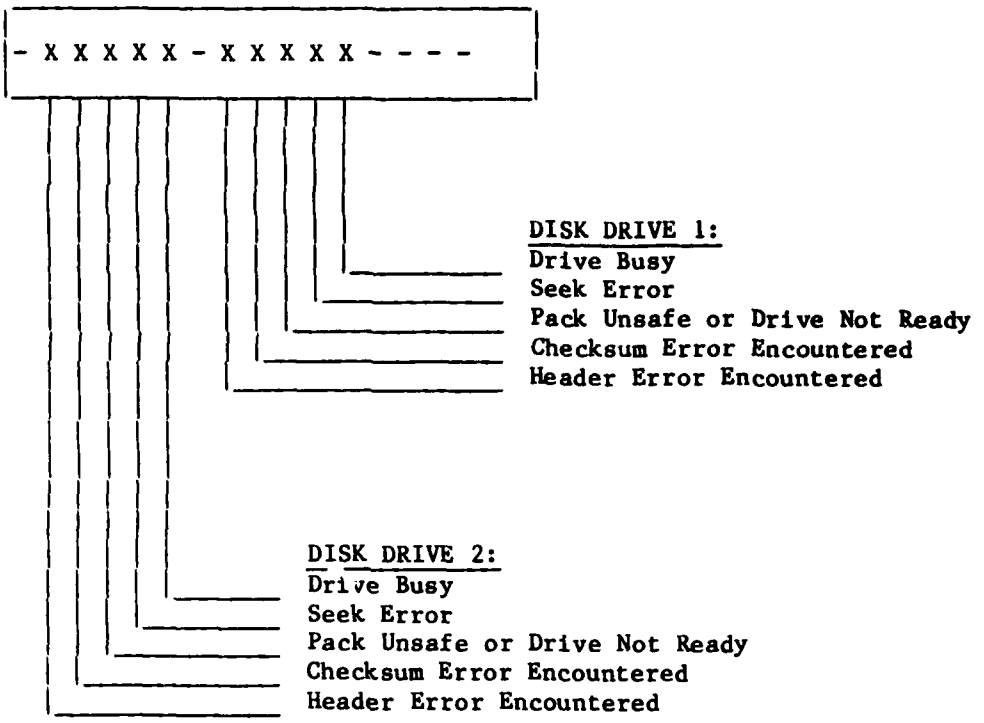


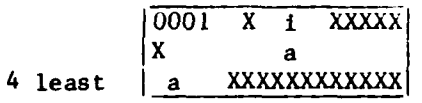
Figure D-4-4. DSS Status Word

D-5.0 COMMANDS

There are 9 commands available to the IAFP and the PDP to transmit to the DSS. They are transmitted in groups of either 3 or 33 16-bit words where the 4 most significant bits of the first word identifies the command. If these bits do not contain a recognizable command, the 3 word grouping is ignored. If the command is recognizable, it is not acted upon until the entire command word grouping is received. For certain commands either 30 16-bit data words or 3 16-bit status words will be transmitted to the interface ring. Table D-5.1 lists the field designators used in describing the formats of the command words.

D-5.1 SET POINTER

This command is a 3 word command. It causes the setting of pointer i to a 19 bit address value. The first word of the 3 command words gives the command and specifies the pointer while the other 2 command words specify the address. If either the value in the pointer field exceeds 39 or the value in the address field exceeds 320K-1, then the 3 word command grouping is ignored. Figure D-5.1 gives the format of the 3 word command.



15 most significant bits of address

4 least significant bits of address

Figure D-5-1. Set Pointer Command Format

TABLE D-5-1. COMMAND FIELD DESIGNATORS

Designator	Number of Bits	Use
a	19	An LCM Address
d	480	40 LCM data words
h	4	A disk drive head
i	6	An index register pointer
j	1	A disk drive id: 0 for disk drive 1, 1 for disk drive 2
m	5	A status word mask for one of the disk drives
t	8	A cylinder id
X	1	A word fill position
CW	12	A track id code word
EM	4	A mask to clear the error code status

D-5.2 WRITE LCM WITH POINTER SPECIFIED

This command is a 33 word command. It causes the writing of 40 words into LCM at the locations referenced by pointer i and to increment the pointer's value by 40. The first command word of the 33 words gives the command and specifies the pointer while the 30 last words provide the data which will be written into LCM. If the value in the pointer field exceeds 39 or if the pointer's value exceeds 320K-41, then the 33 word command grouping is ignored. Figure D-5-2 give the first 4 words of the 33 word command grouping. The remaining 29 words are the rest of the 480 bit data word.

0010 X i XXXXX
XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX
d

Figure D-5-2. Write LCM with Pointer Specified Command Formant

D-5.3 WRITE LCM WITHOUT POINTER SPECIFIED

This command is a 33 word command. It causes the writing of 40 words into LCM at the locations referenced by the pointer that was last specified and to increment the pointer's value by 40. The first of the 33 command words gives the command while the 30 last words provide the data which will be written into LCM. If a pointer has not previously been specified or if the pointer's value exceeds 320K-41, then the 33 word command grouping is ignored. Figure D-5-3 gives the first 4 of the 33 command grouping. The remaining 29 words are the rest of the 480 bit data word.

0011 XXXXXXXXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX
d

Figure D-5-3. Write LCM without Pointer Specified

D-5.4 READ LCM WITH POINTER SPECIFIED

This command is a 3 word command. It causes the reading of 40 LCM words at the locations referenced by pointer i and to increment the pointer's value by 40. The first of the 3 command words gives the command and specifies the pointer. In response to this command, the DSS will transmit to the interface ring 30-16 bit data words which have the 40 LCM words packed into them. If the value in the pointer field exceeds 39 or if the pointer's value exceeds 320K-41, then the 3 word command grouping is ignored. Figure D-5-4 gives the 3 word command grouping.

0100 X i XXXXX
XXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX

Figure D-5-4. Read LCM with Pointer Specified Command Format

D-5.5 READ LCM WITHOUT POINTER SPECIFIED

This command is a 3 word command. It causes the reading of 40 LCM words at the locations referenced by the pointer that was last specified and to increment the pointer's value by 40. The first of the 3 command words gives the command. In response to this command, the DSS will transmit to the interface ring 30 data words which have the 40 LCM words packed into them. If a pointer has not previously been specified or if the pointer's value exceeds 320K-41, then the 3 word command grouping is ignored. Figure D-5-5 gives the 3 word command grouping.

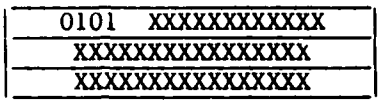


Figure D-5-5. Read LCM without Pointer Specified Command Format

D-5.6 WRITE DISK TRACK

This is a 3 word command. It causes the controlware to write a track of data (3060 bytes or 2040 12-bit LCM words) onto disk j with head t starting at the location referenced by pointer i at cylinder position t and to increment the pointer's value by 2040. The first of the 3 command words gives the command and specifies the pointer, disk drive and head while the second word specifies the cylinder. The third word is a code word which identifies this specific disk track. If either the value in the pointer field exceeds 39, the value in the cylinder field exceeds 199, the value in the head field exceeds 9 or the pointer's value exceeds 320K-2041, then the 3 word command grouping is ignored. Figure D-5-6 give the format of the 3 word command.

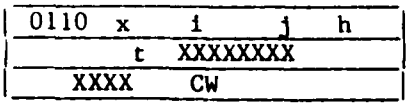


Figure D-5-6. Write Disk Track Command Format

D-5.7 READ DISK TRACK

This is a 3 word command. It causes the controlware to read a track of data (3060 bytes or 2040 12-bit LCM words) of disk j with head t starting at the location referenced by point i at cylinder position t and to increment the pointer's value by 2040. The first of the 3 command words gives the command and specifies the pointer, disk drive and head while the second specifies the cylinder. If either the value in the pointer field exceeds 39, the value in the cylinder field exceeds 199, the value in the head field exceeds 9 or the pointer's value exceeds 320K-2041, then the 3 word command grouping is ignored. Figure D-5-7 gives the format of the 3 word command.

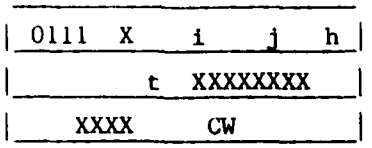


Figure D-5-7. Read Disk Track Command Format

D-5.8 READ STATUS WORD

This is a 3 word command. It causes the current status word to be transmitted to the interface ring. The first of the 3 command words gives the command. Figure D-5-8 gives the format of the 3 word command.

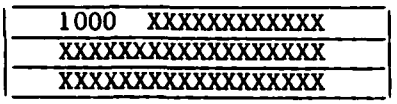


Figure D-5-8. Read Status Word Command Format

D-5.9 SET AND CLEAR STATUS BITS

This is a 1 word command. It causes an exclusive OR operation to occur between the status word bits associated with drive j and the bits in mask m. It also causes an exclusive OR operation to occur between the error code status word and the bits in mask EM. The result of this operation will replace the status word portion associated with drive j.
Figure D-5.9 gives the format of the 3 word command.

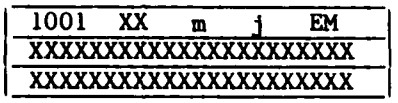


Figure D-5-9. Set and Clear Status Bits Command Format

APPENDIX E

ASSOCIATIVE UNIT MICROCODE
FOR
DEMONSTRATION

This appendix consists of an annotated listing of the Associative Unit microcode which was used in the Advanced Document Retrieval System demonstration.

RETRIEVER ASSOCIATIVE UNIT MICROCODE SUMMARY

ROUTINE	FUNCTION	AU STARTING ADRS	# INSTRUCTIONS
LOAD	Load data	00H	9
SDIR	Segment directory search	10H	12
COMP	Compression lookup	20H	17
SWRD	Stopword search	40H	7
DUMP	Dump	50H	14
			59 *total*

Unused memory was left between routines for potential changes.

The data formats for the demonstration are given in figure E-1.

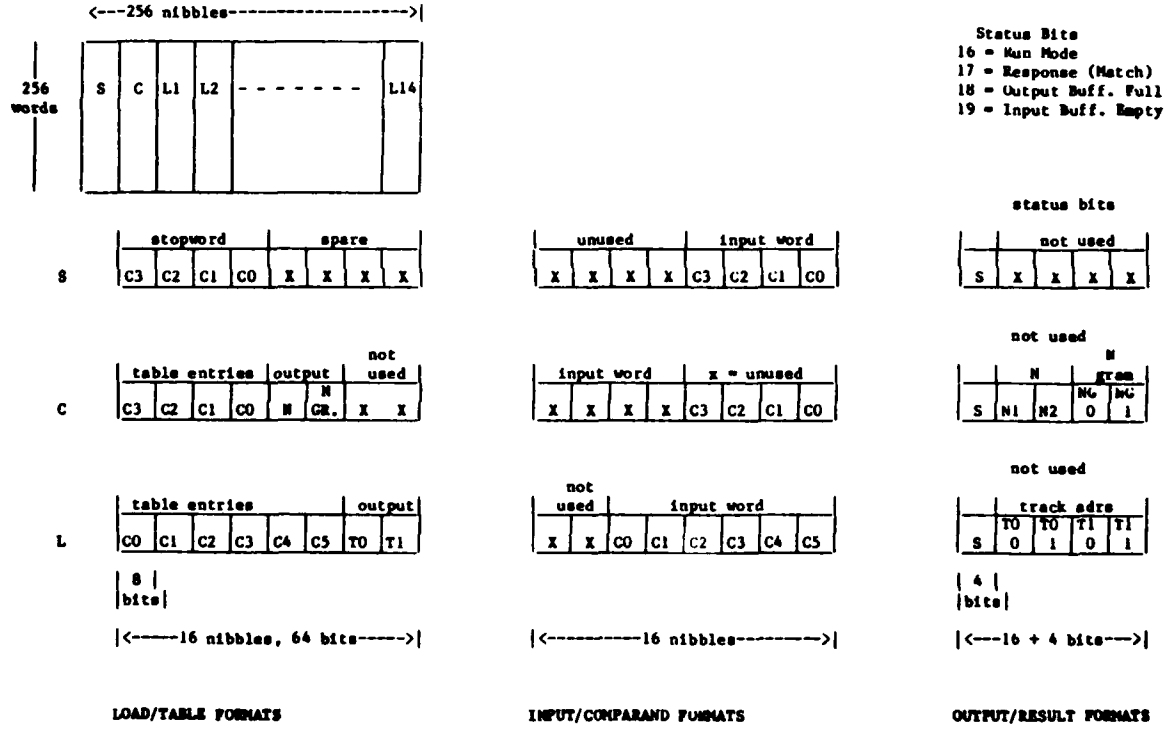


Figure E-1. Retriever AU Data Formats

LOADING A PROGRAM INTO THE AU

The AU microprograms consist of 48 bit words. They are loaded 16 bits at a time as described below.

PROTOCOL

XMAU OP'N	XMAU ADRS	XMAU DATA	
W	0001	0000,0000,0000,0000*	STOP AU (if necessary)
W	0001	0000,0000,0000,PP00	SEND STARTING ADRS PP
W	0004	0000,0000,0000,IIII	FIRST INST, L
W	0004	0000,0000,0000,IIII	FIRST INST, C
W	0004	0000,0000,0000,IIII	FIRST INST, R
W	0004	0000,0000,0000,IIII	SECOND INST, L
W	0004	0000,0000,0000,IIII	SECOND INST, C
W	0004	0000,0000,0000,IIII	SECOND INST, R
			etc.

*only underlined data significant.

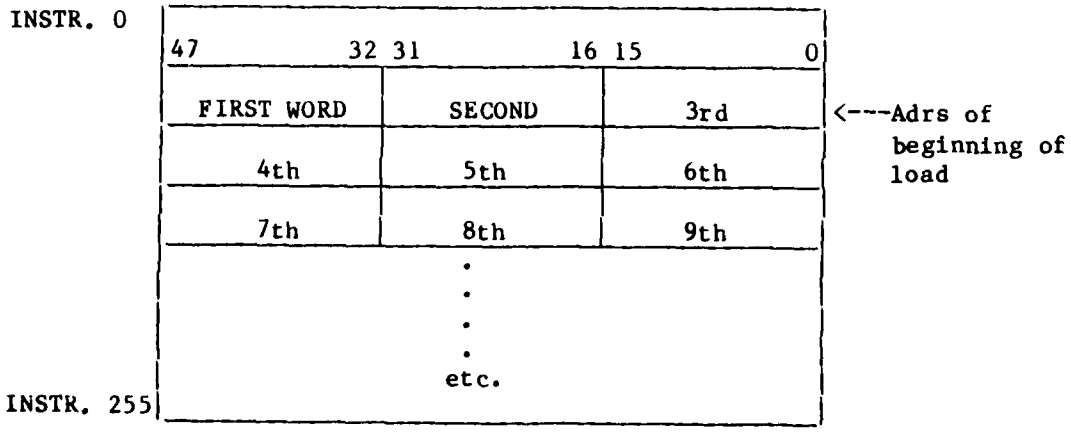


Figure E-2. AU Program Memory Map

LOAD

This routine transfers 256-64 bit data words from AFP to ALE memories. The first data word sent is stored in cell 0 and succeeding words are stored in consecutively higher numbered cells. Exactly 256 data words must be sent. The input data words occupy 16-4 bit columns in ALE memory. The address of the highest column to be occupied must be specified before the transfer.

PROTOCOL

XMAU OP'N*	XMAU ADRS	XMAU DATA	
(W	0001	0000,0000,0000,0000***	STOP AU)**
W	0001	0000,0000,0000,PP01***	START AU PROGRAM @ ADRS PP
W	0000	0000,0000,0000,00HH***	COLUMN ADRS HH
W	0000	IIII,IIII,IIII,IIII	INPUT WORD 1
.	.	.	.
.	.	.	.
.	.	.	.
W	0000	IIII,IIII,IIII,IIII	INPUT WORD 256

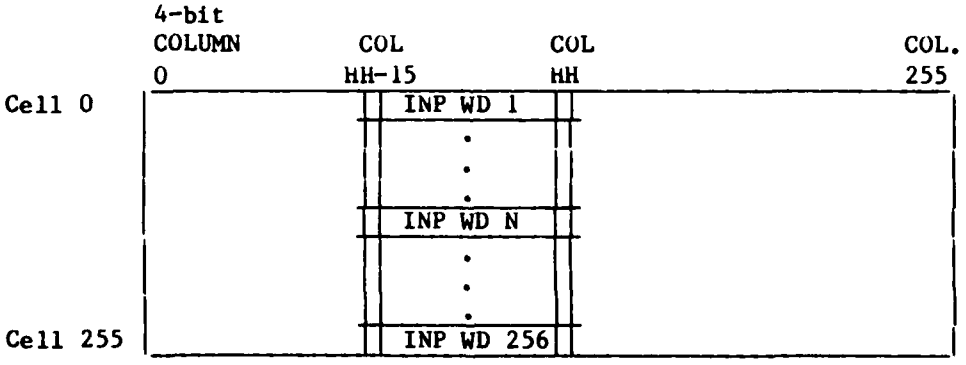


Figure E-3. ALE Memory Map

* After each XMAU write, acknowledge will be delayed until AU is ready for next data word or is done with last word.

** Stop AU function needed if AU state (run/wait) is unknown

*** Only underlined data is significant. Other bits are don't care fields.

LOAD Algorithm Outline

Addr		
0	LOAD:	IF (INPUT RDY) GO TO LOAD.
1		IBUF -> CAR, IBUF -> CTA, 15 -> CTB, SET MARK, SET INPUT RDY.
2	LA:	IF (INPUT RDY) GO TO LA.
3		IBUF -> CDB, WR (F), CTB - 1 -> CTB, RSHIFT IBUF.
4	LB:	RSHIFT IBUF, IBUF -> CDB, CAR - 1 -> CAR, CTB - 1 -> CTB, WR (F), IF (CTB ≠ 0) GO TO LB.
5		CLR MARK (F), CTA -> CAR, SET INPUT RDY.
6		15 -> CTB
7		IF (RESPONSE) GO TO LA.
8		SET WAIT MODE.

LOAD

ADRS	ST	CY	MK	BR	ARR	BB	AB	R	W	M	ALU	IP	CAR	D	I	B	A	O	WT	J	COND	K	
0	0	4		0				0			0	0		0		0		1			2	0	0
1	0	5		0				0			0	1		2		6		0			0	0	F
2	0	4		0				0			0	0		0		0		1			2	0	2
3	D	4		0				2			0	0		5		C		0			0	0	0
4	D	4		0				2			0	5		5		C		1			D	0	4
5	D	6		0				0			0	2		2		0		0			0	0	0
6	0	4		0				0			0	0		0		4		0			0	0	F
7	0	4		0				0			0	0		0		0		1			1	0	2
8	0	4		0				0			0	0		0		0		2			0	0	0

SDIR Segment Directory Search

This routine compares one 48 bit input word with a 48 bit column of data in the ALE memories. The data in the ALE's is stored in ascending order. The AU returns 16 bits of data associated with the greatest column entry that is less than or equal to the input word. The address of the highest column to be searched must be specified. The data and comparand must have the least significant digit rightmost.

PROTOCOL

XMAU OP'N*	XMAU ADRS	XMAU DATA (R/W)
(W	0001	0000,0000,0000,0000*** STOP AU)**
W	0001	0000,0000,0000,PP01*** START AU PROGRAM @ ADRS PP
W	0000	0000,0000,0000,00HH*** COLUMN ADRS HH
W	0000	0000,IIII,IIII,IIII*** INPUT WORD
R	0000	0000,0000,0000,JJJJ*** OUTPUT WORD

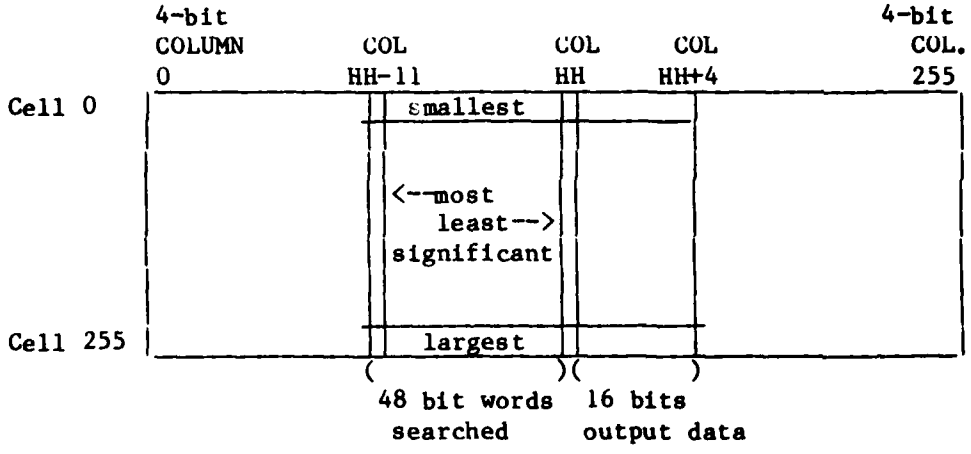


Figure E-4. ALE Memory Map

* After each XMAU operation, acknowledge is delayed until the AU is ready for the next R/W operation or is finished.

** Stop AU function is necessary if the state (Run/Wait) of AU is unknown.

*** Only underlined data is significant.

SDIR Algorithm Outline

Addr		
10	SDIR:	IF (INPUT RDY) GO TO SDIR.
11		IBUF -> CAR, IBUF -> CTA, 11 -> CTB, SET MARK, SET INPUT RDY, SET CARRY.
12	SDA:	IF (INPUT RDY) GO TO SDA.
13		IBUF -> CDB, ALU = A - B, COUT -> CARRY, CTB - 1 -> CTB, RSHIFT IBUF.
14	SDB:	RSHIFT IBUF, IBUF -> CDB, ALU = A - B, COUT -> CARRY, CTB - 1 -> CTB, CAR - 1 -> CAR, IF (CTB ≠ 0) GO TO SDB.
15		CLR MARK IF CARRY = 1, CTA -> CAR, CLR CARRY IF CARRY = 1.
16		SET MARK IF CELL BELOW MARKED, CAR + 1 -> CAR, ALU = BBUS, BBUS = RAM, RBUS = ALU, LOAD RREG.
17		CAR + 1 -> CAR, ALU = BBUS, BBUS = RAM, RBUS = ALU, LOAD RREG.
18		LOAD OUTPUT BUFFER, CAR + 1 -> CAR, ALU = BBUS, BBUS = RAM, RBUS = ALU, LOAD RREG.
19		LOAD OUTPUT BUFFER, CAR + 1 -> CAR, ALU = BBUS, BBUS = RAM, RBUS = ALU, LOAD RREG.
1A		LOAD OUTPUT BUFFER, SET INPUT RDY.
1B		LOAD OUTPUT BUFFER, SET OUTPUT RDY, SET WAIT MODE.

SDIR

ADRS	ST	CY	MK	BR	ARB	BB	ABB	R	W	M	ALU	IP	CAR	D	I	B	A	O	WT	J	COND	K	
10	0	0		0				0			0	0		0		0		1			2	1	0
11	0	9		0				0			0	1		2		6		0			0	1	1
12	0	0		0				0			0	0		0		0		1			2	1	2
13	0	C		0				0			6	0		5		C		0			0	0	0
14	0	C		0				0			6	5		5		C		1			D	1	4
15	3	9		0				0			0	2		0		0		0			0	0	0
16	7	1		0				5			A	4		0		0		0			0	0	0
17	0	0		0				5			A	4		0		0		0			0	0	0
18	0	0		0				5			A	4		0		0		4			0	0	0
19	0	0		0				5			A	4		0		0		4			0	0	0
1A	0	0		0				0			0	0		2		0		4			0	0	0
1B	0	0		0				0			0	0		0		0		E			0	0	0

COMP Compression Lookup

This routine performs a text compression table look-up algorithm using a 64 bit input word containing four characters. The data in the ALE's is stored in ascending order with the most significant character on the right. The input word must contain the most significant character on the right. The AU returns a 16 bit result associated with the column entry selected by the algorithm. The compression look-up algorithm is based on a tech. memo by W. R. Cyre. The address of the highest column to be searched must be specified.

INPUT WORD FORMAT: XXXX,XXXX,TTSS,RRQQ

QQ - first character (most significant)

RR - second character

SS - third character

TT - fourth character

PROTOCOL

XMAU OP'N*	XMAU ADRS	XMAU DATA
(W	0001	0000,0000,0000,0000*** STOP AU)**
W	0001	0000,0000,0000,PP01*** START AU PROGRAM @ ADRS PD
W	0000	0000,0000,0000,00HH*** COLUMN ADRS HH
W	0000	0000,0000,TTSS,RRQQ*** INPUT WORD
R	0000	0000,0000,0000,JJJJ*** OUTPUT WORD

* After each XMAU operation, the acknowledge to the AFP is delayed until the AU is ready for the next R/W operation or is finished.

** Stop AU function is required if the state (Run/Wait) of AU is unknown.

*** Only underlined data is significant.

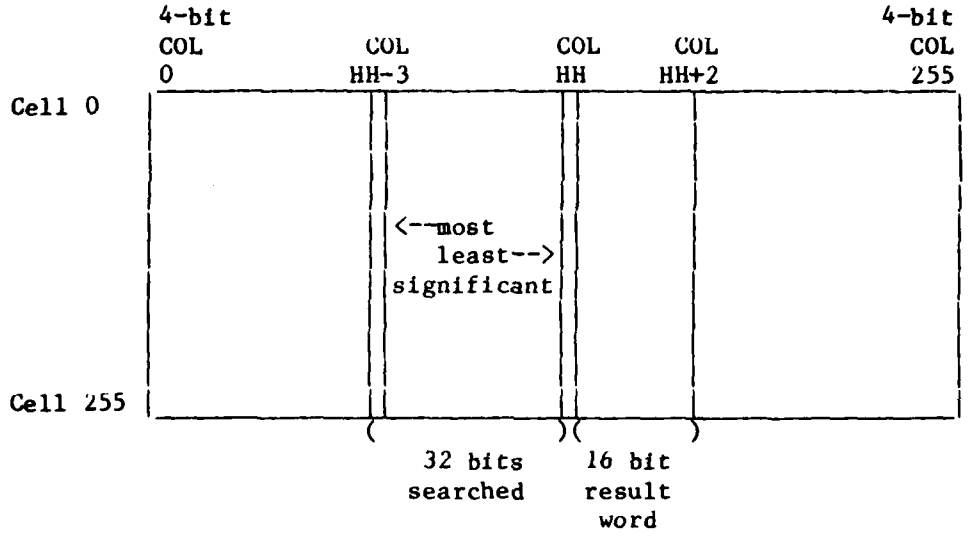


Figure E-5. COMP - ALE Memory Map

COMP Algorithm Outline

Addr		
20	COMP:	IF (INPUT RDY) GO TO COMP.
21		IBUF -> CAR, IBUF -> CTA, SET MARK, SET CARRY, SET INPUT RDY, 3 -> CTB.
22	CA:	IF (INPUT RDY) GO TO CA.
23		IBUF -> CDB, RSHIFT IBUF, BBUS = RAM, ABUS = CDB, ALU = $\overline{A \oplus B}$, CLR MARK IF ($\overline{W}$), CLR CARRY IF ($\overline{W}$).
24		IBUF -> CDB, RSHIFT IBUF, BBUS = RAM, ABUS = CDB, ALU = $\overline{A \oplus B}$, CLR MARK IF ($\overline{W}$), CLR CARRY IF ($\overline{W}$), CAR - 1 -> CAR.
25	CB:	RSHIFT IBUF, IBUF -> CDB, BBUS = RAM, ABUS = CDB, ALU = $\overline{A \oplus B}$, CLR MARK IF ($\overline{W}$), CAR - 1 -> CAR.
26		RSHIFT IBUF, IBUF -> CDB, BBUS = RAM, ABUS = CDB, ALU = $\overline{A \oplus B}$, CLR MARK IF ($\overline{W}$), CAR - 1 -> CAR.
27		CTB - 1 -> CTB, BBUS = RAM, ALU = $\overline{B}$ IF ($\overline{W}$), CLR CARRY.
28		CAR + 1 -> CAR, BBUS = RAM, ALU = $\overline{B}$ IF ($\overline{W}$), CLR CARRY.
29		CAR - 1 -> CAR, IF (CARRY) SET MARK
2A		IF (MARK) SET CARRY, IF (CTB $\neq$ 0) GO TO 25.
2B		CTA -> CAR.
2C		SET INPUT RDY, CAR + 1 -> CAR, ALU = BBUS, BBUS = RAM, BBUS = ALU, LOAD RREG.
2D		CAR + 1 -> CAR, ALU = BBUS, BBUS = RAM, RBUS = ALU, LOAD RREG.
2E		LOAD OUTPUT BUFFER, CAR + 1 -> CAR, ALU = BBUS, BBUS = RAM, RBUS = ALU, LOAD RREG.
2F		LOAD OUTPUT BUFFER, CAR + 1 -> CAR, ALU = BBUS, BBUS = RAM, RBUS = ALU, LOAD RREG.
30		LOAD OUTPUT BUFFER.
31		LOAD OUTPUT BUFFER, SET OUTPUT RDY, SET WAIT MODE.

COMP

ADRS	ST	CY	MK	BR	ARB	BB	AB	R	W	M	ALU	IP	CAR	D	I	B	A	O	WT	J	COND	K	
20	0	0		0				0			0	0		0		0		1			2	2	0
21	0	9		0				0			0	1		2		6		0			0	0	3
22	0	0		0				0			0	0		0		0		1			2	2	2
23	A	6		0				1			9	0		5		0		0			0	0	0
24	A	6		0				1			9	5		5		0		0			0	0	0
25	A	2		0				1			9	5		5		0		0			0	0	0
26	A	2		0				1			9	5		5		0		0			0	0	0
27	A	4		0				1			5	0		0		C		0			0	0	0
28	A	4		0				1			5	4		0		0		0			0	0	0
29	3	1		0				0			0	5		0		0		0			0	0	0
2A	1	8		0				0			0	0		0		0		1			D	2	5
2B	0	0		0				0			0	2		0		0		0			0	0	0
2C	0	0		0				5			A	4		2		0		0			0	0	0
2D	0	0		0				5			A	4		0		0		0			0	0	0
2E	0	0		0				5			A	4		0		0		4			0	0	0
2F	0	0		0				5			A	4		0		0		4			0	0	0
2G	0	0		0				0			0	0		0		0		4			0	0	0
30	0	0		0				0			0	0		0		0		E			0	0	0

SWRD Stopword Search

This routine compares one 32 bit input word for exact match with a 32 bit column of data in ALE memories. Bit 17 of the output word is set if a match is found, otherwise it is cleared. The address of the highest column of the compare must be specified.

PROTOCOL

XMAU OP'N*	XMAU ADRS	XMAU DATA (R/W)
(W	0001	0000,0000,0000,0000*** STOP AU)**
W	0001	0000,0000,0000,PP01*** START AU PROGRAM @ ADRS PP
W	0000	0000,0000,0000,00HH*** COLUMN ADRS HH
W	0000	0000,0000,IIII,IIII*** INPUT WORD
R	0000	0000,0000,000M,0000*** OUTPUT WORD

C_H = no match

E_H = match

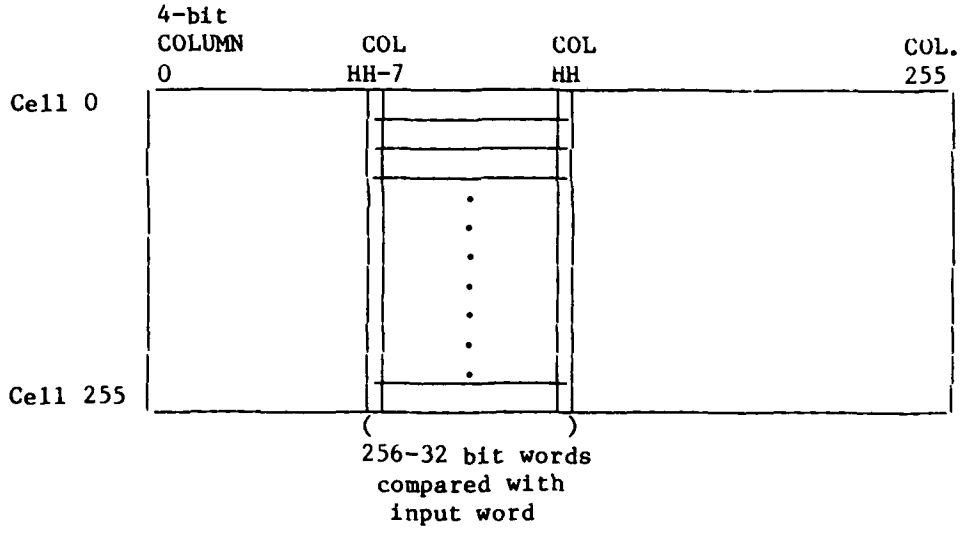


Figure E-6. SWRD - ALE Memory Map

* After each XMAU write operation, acknowledge is delayed until the AU is ready for the next R/W operation or is finished with its program.

** Stop AU function needed if AU state (Run/Wait) is unknown.

*** Only underlined data is significant.

SWRD Algorithm Outline

Addr		
40	SWRD:	IF (INPUT RDY) GO TO SWRD.
41		IBUF -> CAR, SET MARK, SET INPUT RDY, 7 -> CTB.
42	SA:	IF (INPUT RDY) GO TO SA.
43		IBUF -> CDR, CTB - 1 -> CTB, ABUS = CDB, BBUS = RAM, ALU = $\overline{A \oplus B}$, CLR MARK ($\overline{W}$), RSHIFT IBUF.
44	SB:	RSHIFT IBUF, IBUF -> CDR, ABUS = CDB, BBUS = RAM, ALU = $\overline{A \oplus B}$, CLR MARK ($\overline{W}$), CAR - 1 -> CAR, CTB - 1 -> CTB, IF (CTB $\neq$ 0) GO TO SB.
45		NOP.
46		SET INPUT RDY, SET OUTPUT RDY, SET WAIT MODE.

SWRD

ADRS	ST	CY	MK	BR	ARB	BB	AB	R	W	M	ALU	IP	CAR	D	I	B	A	O	WT	J	COND	X	
40	0	0		0		0		0			0	0		0		0		1			2	4	0
41	0	1		0		0		0			0	1		2		4		0			0	0	7
42	0	0		0		0		0			0	0		0		0		1			2	4	2
43	A	2		0		1		9			0	5		C		0		0			0	4	0
44	A	2		0		1		9			5	5		C		1		D			4	4	
45	0	0		0		0		0			0	0		0		0		0			0	0	0
46	0	0		0		0		0			0	0		2		0		A			0	0	0

DUMP

This routine transfers a 64 bit column of data from the ALE memories to the AFP, 16 bits at a time. The AFP must send the lowest column address to be transferred, then read 1024 - 16 bit quantities.

PROTOCOL

XMAU OP'N*	XMAU ADRS	XMAU DATA
(W	0001	0000,0000,0000,0000*** STOP AU)**
W	0001	0000,0000,0000,PP01*** START AU PROGRAM @ ADRS PP
W	0000	0000,0000,0000,00HH*** COLUMN ADRS HH
R	0000	0000,0000,0000,JJJJ*** OUTPUT WORD 1
.	.	. .
.	.	. .
.	.	. .
R	0000	0000,0000,0000,JJJJ*** OUTPUT WORD 1024

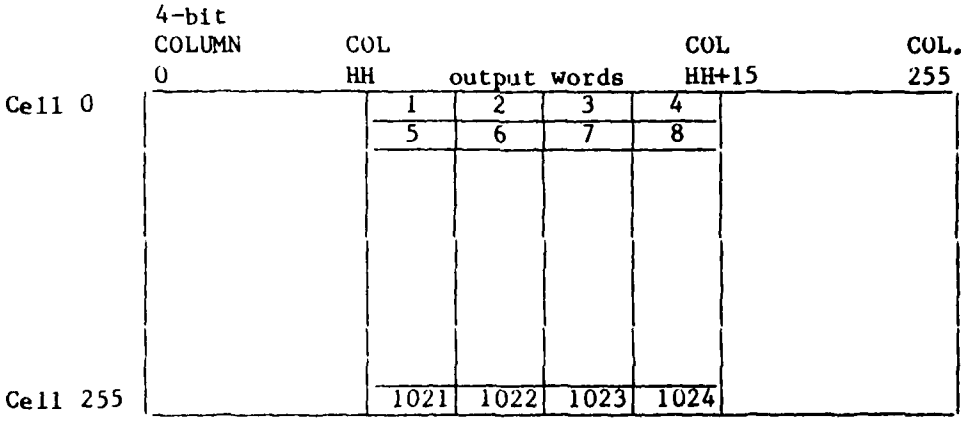


Figure E-7. DUMP ALE Memory Map

* After each XMAU operation, acknowledge is delayed until AU is ready for next transfer or is done.
** Stop AU function required if AU status (Run/Wait) is unknown.
*** Only underlined data is significant. Other bits are don't care fields.

DUMP Algorithm Outline

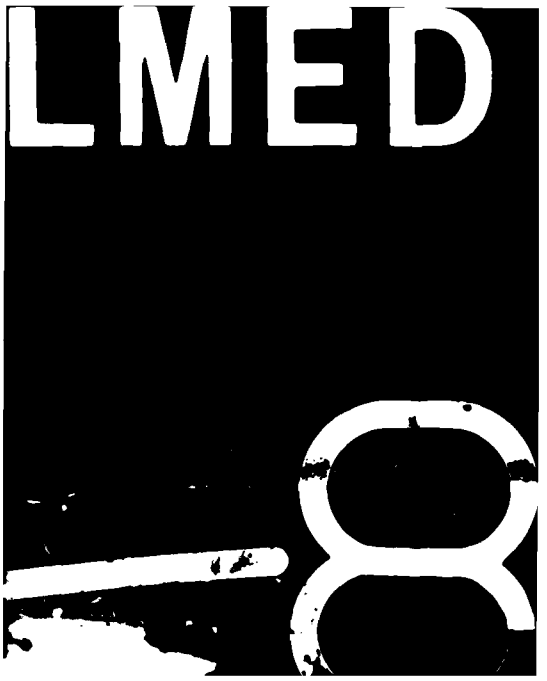
Addr		
50	DUMP:	IF (INPUT RDY) GO TO DUMP.
51		IBUF -> CAR, IBUF -> CTA, SET INPUT RDY, SET MARK.
52	DA:	4 -> CTB.
53	DB:	BBUS = RAM, ALU = B, RBUS = ALU, LOAD RREG.
54		BBUS = RAM, ALU = B, RBUS = ALU, LOAD RREG, CAR + 1 -> CAR.
55		BBUS = RAM, ALU = B, RBUS = ALU, LOAD RREG, LOAD OBUF, CAR + 1 -> CAR.
56		BBUS = RAM, ALU = B, RBUS = ALU, LOAD RREG, LOAD OBUF, CAR + 1 -> CAR.
57		LOAD OBUF, CAR + 1 -> CAR.
58		LOAD OBUF, SET OUTPUT RDY, CTB - 1 -> CTB.
59	DC:	IF (OUTPUT RDY) go to DC.
5A		IF (CTB ≠ 0) GO TO DB.
5B		CLR MARK IF FIRST MARKED, CTA -> CAR.
5C		NOP
5D		IF (RESPONSE) GO TO DA.
5E		SET WAIT MODE.

DUMP

ADRS	ST	CY	MK	BR	ARB	BB	AB	R	W	M	ALU	IP	CAR	D	I	B	A	O	WT	J	COND	K	
50	0	0		0				0			0	0		0		0		1			2	5	0
51	0	1		0				0			0	1		2		2		0			0	0	
52	0	0		0				0			0	0		0		4		0			0	0	4
53	0	0		0				5			A	0		0		0		0			0	0	0
54	0	0		0				5			A	4		0		0		0			0	0	0
55	0	0		0				5			A	4		0		0		4			0	0	0
56	0	0		0				5			A	4		0		0		4			0	0	0
57	0	0		0				0			0	4		0		0		4			0	0	0
58	0	0		0				0			0	0		0		0		C			C	0	0
59	0	0		0				0			0	0		0		0		1			3	5	9
5A	0	0		0				0			0	0		0		0		1			D	5	3
5B	D	2		0				0			0	2		0		0		0			0	0	0
5C	0	0		0				0			0	0		0		0		0			0	0	0
5D	0	0		0				0			0	0		0		0		1			1	5	2
5E	0	0		0				0			0	0		0		0		2			0	0	0

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


LMED