AD NUMBER

AD489661

NEW LIMITATION CHANGE

TO

Approved for public release, distribution unlimited

FROM

Distribution authorized to U.S. Gov't. agencies and their contractors; Administrative/Operational Use; Jul 1966. Other requests shall be referred to Rome Air Development Center, Griffiss AFB, NY.

AUTHORITY

RADC USAF ltr, 17 Sep 1971

RADC-TR-65-397, Vol II
FINAL REPORT

ANALYSIS OF SMALL ASSOCIATIVE MEMORIES FOR DATA STORAGE AND RETRIEVAL SYSTEMS

VOLUME II. TECHNICAL DISCUSSION

Robert S. Green, Dr. Jack Minker, and Warren E. Shindle

TECHNICAL REPORT NO. RADC-TR-65-397, Vol II

July 1966

This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMLI), GAFB, N.Y. 13440.

Rome Air Development Center
Research and Technology Division
Air Force Systems Command
Griffiss Air Force Base, New York

ANALYSIS OF SMALL ASSOCIATIVE MEMORIES FOR DATA STORAGE AND RETRIEVAL SYSTEMS

VOLUME II. TECHNICAL DISCUSSION

Robert S. Green, Dr. Jack Minker, and Warren E. Shindle

This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMLI), GAFB, N.Y. 13440.

MAINLINE ENGINEERING, MAPLE SHADE, N.J.
9/26/66 [illegible]

FOREWORD

This report was prepared by the Auerbach Corp., Philadelphia, Pa.; under Contract Number AF30(602)-3564; Project Number 4594 and Task Number 459402. The RADC project engineer is Ronald J. Ferris, EMIID.

This is a final report and covers the period of work from October 1964 to September 1965; the originator's report number is 1231-TR2.

The results of this study are contained in two volumes: Volume I presents in summary form, an introduction, findings, conclusions, and recommendations. In addition, this volume contains a discussion on associative memories and the comparative analyses of the associative memory configurations studied. A short critique of the Goodyear Associative Processor and a bibliography are also given in this volume.

Volume II contains the detailed discussion of the study. This volume specifies the problem of "how" the study was conducted. It presents the technical details, the relevant flowcharts, algorithms, and encoding. Each section of this volume contains information pertinent to the approach, technical details, and the findings and conclusions. Supplementary information relevant to the study is contained in the appendices.

The principal investigators on this study were Mr. James Dugan, Mr. Robert Shaw Green (Project Engineer - hardware related studies), Mrs. Renee Jasper, Mr. Marvin Katz, Dr. Jack Minker (Program Manager), Mr. James Mulford, and Mr. Warren E. Shindle (Project Engineer - problem analysis and programming related studies). The authors would like to express their appreciation to the numerous members of the AUERBACH technical staff who were called upon for expert advice on specialized areas of study. These are: Mr. George Neborak, Mr. Sheldon Einhorn, Mr. Don Russel, Mr. Ken Rose, Mr. Lawrence Feidelman, Mr. Ralph Howe, Mr. Arthur Hughes, Mr. Stephen Leibholz, Mr. Lester Probst, Mr. Robert Rossheim, and Dr. Jerome Sable.

Release of subject report to the general public is prohibited by the Strategic Trade Control Program, Mutual Defense Assistance Control List (revised 6 January 65), published by the Department of State.

This technical report has been reviewed and is approved.

Approved: Howard Davis
ROBERT J. QUINN, JR.
Colonel, USAF
Chief, Intel and Info
Processing Div

Approved:
FRANK J. TOMAINI
Chief, Info Processing Branch

FOR THE COMMANDER:
IRVING J. GABELMAN
Chief, Advanced Studies Group

TABLE OF CONTENTS
(Summary of Volumes I and II)

VOLUME I
Management Report

VOLUME II
Technical Discussion

TABLE OF CONTENTS, VOLUME II

TABLE OF CONTENTS, VOLUME II (Cont.)

TABLE OF CONTENTS, VOLUME II (Cont.)

LIST OF ILLUSTRATIONS, VOLUME II

LIST OF ILLUSTRATIONS, VOLUMI (Cont.)

LIST OF TABLES, VOLUME II

LIST OF TABLES, VOLUME II (Cont.)

EVALUATION

I. Study Objective:

The objective of this work was to determine the value of state-of-the-art associative memories to more efficient execution of typical non-numeric and intelligence problems by the general purpose computer. Thus it was not the intent to provide an absolute figure of merit which would set the value of small associative memories. Nor was it the intent to evaluate the full range of memory logic design or nature of connection to the computer which an associative memory device may take. It was hoped however, that a useful reference point would be established with which to guide further studies. It should be noted that these studies are the first significant attempts to evaluate an associative memory in a system context; * i.e., the associative memory is considered in association with a general purpose computer, typical peripheral hardware, and system programming.

II. Approach:

A. Selection of CDC 1604B and CDC 818 Disc

At the outset it was realized that the CDC 1604B computer, the CDC 818 disc and the Goodyear associative processor might not result in an optimum combination for the problems selected. Rather, the selection was based on the following:

1. The CDC 1604B is representative of "second generation" computers which are the most prominent in Air Force use today. Moreover, due to the high cost of replacing computer systems, it may be some time before present systems are replaced. Consequently, considerable savings might be realized from determining how these computers could be used more efficiently.

2. RADC recognized that there are several factors in addition to the choice of an associative memory and a general purpose computer which affect study results. These factors are associated with the software and with file structure, query statements, etc. thus by selecting the CDC 1604B and the CDC 818 disc, we provide the capability to modify the software and gain a more thorough understanding of the role of software in evaluating associative memories.

*See Mr. Ronald Ferris' comments in Volume II of the International Federation for Information Processing 1965 proceedings under the panel session on Content Addressable Memories.

3. Since the hardware is available at RADC, the software programs developed can be implemented and exercised in order to provide empirical data with a minimum of cost and time.

It should be emphasized that the selection of the CDC 1604B computer and CDC 818 disc does not affect the accuracy of the comparison made between the associative memory configuration and the non-associative memory configuration, since in both cases the same general purpose computer and disc were used. In fact, this is the type of data that is of interest to us.

B. Selection of Goodyear Associative Processor (GAP)

The primary intent of this effort was to evaluate the GAP merely as a first step in a long range plan to evaluate associative memories. Consequently, whether or not the GAP is optimally designed for a specific task is relatively unimportant. Instead it should be viewed as a research tool for use in obtaining valuable information pertaining to associative processing, particularly in the area of software. Accordingly the GAP design was selected with the intent to evaluate a state-of-the-art product with reasonable cost. Finally, it should be emphasized that evaluation of GAP cannot be construed as an evaluation of Goodyear Aerospace's capability to design computers.

C. Selection of Problems

The problems used throughout the study are actually processing functions. The intent was to choose a set of processing functions common to many specific problems and in so doing obtain generalizable results. Hence, the term sea surveillance merely indicates the source of the processing functions employed.

The selection of problems is a very important step and should not be taken lightly. The approach utilized in this study was briefly the following: *

1. Identify a class of problems within the Air Force

2. Identify the processing functions within that class and select the most common.

3. Based on a set of criteria; i.e., priority, availability of data base, etc. a final set of processing functions were selected. The criteria used in this analysis is as follows:

*This work was performed in-the-house at RADC. The details of this work are contained in a report located at RADC.

a. Does the product satisfy an operational requirement?

(1) Has a machine program been written to produce it?
(2) How many producers use it?
(3) Has the data base been built?

b. Are the program steps (operations, algorithm, etc.) common to other programs?

c. Is the program and data base description accessable?

(1) Is it releasable to contractor (Security) ?
(2) How soon can it be delivered to contractor (where do we get it and must it be sanitized) ?

d. Has program been written for 1604B class machine, i.e., is it one of these: 7090, 1410, 1218?

D. Evaluation Criteria and Measures

The following criteria and measures were developed by RADC, Auerbach Corporation and Goodyear Aerospace Corporation for use in their respective investigations:

Criteria	Measures
A. Complexity of Programming Required	1. No. of Instructions in program.
	2. Total time and cost of program.
	3. Time and cost of producing general flow charts (problem analysis & definition).
	4. Time and cost of detailed flow charts (required for time analysis).
B. Accuracy of Programming	1. No. of errors listed in (A3) and (A4) above.
	2. Type of errors located in (A3) and (A4) above.
C. Complexity of Hardware Required (AM)	1. Type and number of components used in AM e.g.; 100 transistors.
	2. Cost of AM and necessary interface.
	3. Time and cost for each AM macro instruction.
	4. Time and cost for each basic AM instruction.
	5. No. of AM searches required to complete each AM macro & basic instruction.

Criteria (Cont.)	Measures (Cont.)
D. Reliability of Hardware	1. Reliability of each component multiplied by some factors, e.g., no. of components.
E. System Efficiency	1. Time to execute program (machine run time). 2. Total cost of programming, hardware, and machine running time. 3. No. of AM & 1604B instructions used (and not used) and frequency of each. 4. Amount of data that is read into and out of AM and frequency of transfers. 5. Time AM is idle. 6. Time required for 1604B to set up AM processing (data transfers etc.). 7. Ratio of storage requirements vs availabilities. 8. Time 1604B is idle. 9. Time disc is idle. N/A 10. % of relative running & idle times for the 1604B, Disc, & AM.

Key

N/A - Not Applicable to Goodyear

III. Use of Study Results:

The results of this study provide a good reference point when determining the usefulness of proposed associative memories. It is believed that this type of investigation, i.e., within a system context is a most important and necessary step. It is our hope that workers in the associative memory field can use this and future studies as a yardstick in designing associative memory - general purpose computer configurations. RADC encourages a close examination of criteria and measures to be used in evaluation of associative memories. We also encourage those interested to examine this effort critically and would make welcome suggestions.

SECTION I. INTRODUCTION

During the past few years a great deal of speculation has arisen concerning the effectiveness of associative memories, both large and small, within a computing system. The concept of an associative memory in which data is accessed by content rather than by its physical address has had considerable intuitive appeal for those interested in data manipulation rather than computational problems.

The technology associated with implementing such devices has made some progress. Several small associative memories have been implemented which contain approximately 2,048 words of memory and, depending upon the device, each word has 48 bits. Such memories have been constructed from so-called BILOC, BICORE, or similar devices. This technology is rather limited with respect to the core sizes that can be achieved at a reasonable cost. The technology associated with large memories is, on the other hand, being developed on the research level. Superconductive devices are being utilized in this technology.

Although considerable attention has been paid to the hardware aspects for both large and small associative memories (AM), several other aspects of importance have not received adequate attention. Thus, if one reviews the extensive literature as noted in the bibliography contained in Volume I of this report, very little may be found which describes the manner in which an AM can be integrated within a computer system, the programming tools that are required to take effective advantage of the memory, or the advantage or disadvantage of such memories in specific applications. Considering the early state of this growing technology, this is not surprising.

Rome Air Development Center (RADC) has developed a comprehensive plan to determine the advantages and disadvantages of both large and small associative memories. The work reported by the AUERBACH Corporation in these volumes is part of this comprehensive plan. Specifically, the study has focused attention upon small associative memories as typified by the Goodyear Associative Memory (a description of the Goodyear Associative Processor (GAP) is contained in Appendix A).

RADC has requested that the AUERBACH Corporation investigate a total data storage and retrieval system to determine the effectiveness of a so-called hybrid associative computer system for such a problem, and to answer some of the following questions:

(1) What is the most effective hybrid system?

(2) How can an associative memory be integrated best into the computer system?

(3) What are the costs associated with developing such devices?

(4) What possible variants should be considered of the GAP technology which might enhance the AM or decrease the cost with little effect on the processing?

(5) How effective are the variant systems?

(6) Will any of the AMs affect the overall processing in the various systems enough to warrant its use.

(7) Is a small associative memory effective?

(8) What is the influence of an AM on flowcharting conventions and on programming?

Answers to most of these questions are found throughout this volume. Volume I, the management report, provides the answers and limitations upon the answers as well as references to Volume II in which the detailed support is provided for the general statements.

1.1 OVERALL SCOPE OF RADC ASSOCIATIVE MEMORY PROGRAM

The larger goal of the investigation is to aid RADC in the evaluation of the applicability and usefulness of a class of small associative memories available in the present state of the technology. This study is only one part of a larger and more comprehensive effort being directed by RADC, and while this investigation has a clearly defined and self-contained goal, the larger context must be considered. Insights and specific results and conclusions of this study are reported to RADC whether they are of primary importance to the fulfillment of the contract or a secondary result of the study.

Within the framework of the larger RADC effort, the overall goal of this investigation may be stated: Evaluate the applicability, effectiveness, and efficiency of hybrid associative processors. The evaluation is to be performed by comparing associatively oriented systems against the best conventional systems employing otherwise comparable equipment. A sufficient range of hybrid organizations and depth of analysis is to be included to ensure a concrete and positive evaluation. The investigation is to consider both programming and equipment factors.

Within this context, decisions have been made affecting the specific scope methodology, and its relationship to other research projects sponsored by RADC.

1.2 STUDY CONSIDERATIONS

1.2.1 Approach Considerations

Several approaches can be taken in investigating the effectiveness of small associative memories. Each approach has certain advantages and disadvantages.

One approach widely used to evaluate alternative conventional equipment is to select several complicated problems and to code these problems in detail for the alternative configurations. Advantages to such an approach are:

(1) A large number of manageable problems can be defined, detailed, and analyzed.

(2) Alternative situations can be considered; e.g., alternative data structures, and record and file organizations may be considered.

(3) Alternative problems bearing upon several subject areas could be considered, e.g., compiler developments, language developments, and data retrieval.

Disadvantages attendant with such an approach are:

(1) The importance that one should attach to each individual problem is difficult to define since its frequency of use in a particular system may be difficult to assess.

(2) Problems of importance to specific areas of interest on a particular problem may not have been covered.

A second approach is to select a single comprehensive problem, potentially rich in associative memory operations for study. This approach has several advantages:

(1) A comprehensive system is viewed for all data processing functions so that the frequency of operations is known and no processes are overlooked.

(2) Areas which may not have been selected for associative memory operations may be uncovered by being forced to review all processes.

(3) Assumptions as to the location of data which may tend to bias a solution towards the associative memory is avoided since one must take cognizance of the precise location of data as developed in previous processing steps.

(4) The associative memory can be utilized to store more than one class of data as required by dynamic system processing.

As with the previous approach, there are certain inherent disadvantages. Some of these are:

(1) The solution reached for a particular problem may be specific and may preclude generalization to other problems.

(2) The specific problem may be such that the full extent of the AM is not utilized.

(3) Alternative solutions might not be evaluated sufficiently since sufficient time for each alternative might not be possible.

(4) Excessive time might be spent in systems design of the problem taking time away from study of the details of associative memory processing.

Realizing the advantages and disadvantages attendant with each approach, RADC selected the latter for the AUERBACH Corporation and the former for other contractors. The study realized all of the advantages in this approach. By being forced to think at both a systems and a detailed level, many observations concerning associative memory processing were developed. At the same time, several of the above disadvantages are pertinent. The solution reached is believed to be sufficiently general and applicable to problems of the same class. However, the queries and maintenance operations specified as "typical" in the sea surveillance problem did not sufficiently exercise the AMs studied. That is, the queries were very simple and did not require much processing while the data resided in the associative memory. More complex questions could have shown the AM to greater advantage. Alternative solutions (e.g., different file structures) could not be studied. Thus, perhaps a more clever file structure than that utilized in this study could have been devised which would have shown the AMs to greater advantage. The authors cannot assess this since sufficient time did not exist for exploring this possibility. They can verify that some time was spent in systems design which could have been spent more profitably on the details of AM programming. In spite of this, much was learned about the organization of a system around an AM, the effect of an AM upon system organization, and flowcharting.

The study team believes that, in the main, the advantages of the approach outweighed the disadvantages. A large amount of information of considerable interest and importance to associative memory technology has been developed and is reported in these volumes.

1.2.1.1 Hardware Approach. As representative of the current technology of small associative memories, the Goodyear Associative Processor was selected for study.

In addition to studying the Goodyear Associative Processor in detail, both from the hardware and software viewpoints, several variant AMs based upon the Goodyear technology were to be studied. There were several AM variations that were of particular interest to the study.

(1) Logical Organization of the AM: The range of variation desired was to investigate a minimal control unit requiring detailed control by instruction from the 1604-B and one which was independent of the 1604-B.

(2) Interface with the 1604-B. The variations to be considered ranged from one which required no modifica- to the 1604-B to one which would have a special channel and special register to one fully integrated with the 1604-B.

(3) Instruction Set. The variations would be from a minimal set which would decrease costs of the AM to one in which extensive search capabilities generalizing the GAP would be developed.

The selection of the specific memories studied were developed after extensive meetings between the programmers and the hardware designers. The details concerning the specific memories chosen are presented in Section II of this report.

1.2.1.2 Cost Analysis. Costing played a major role in the selection of the specific variants that were to be exercised in the study. To determine the costs involved, the approach utilized was to study GAP thoroughly, develop logic diagrams, and estimate the amount of equipment that would be required. AUERBACH's knowledge of the state of the art of integrated circuit technology and BILOC devices was utilized. The approach was to develop cost factors to permit other hardware variations to be costed. Wherever the cost analysis was to show a minor increase in cost to implement a particular capability desired by a programmer , the feature was added. Cost analysis information is provided in Section II.

1.2.1.3 Programming. The problem to be programmed was to be effected in a general manner. The nature of intelligence problems is such that if a system were based upon the specific requirements, other requirements would arise requiring further programming. The approach was, therefore, to permit arbitrary logical conditions to be specified to the resulting system. This approach avoids biasing programs to handle very specific tasks, which potentially have little utility thereafter.

In addition to considering the Goodyear Associative Processor and the variant associative memories, a non-associative memory configuration consisting of the system without an AM was to be evaluated.

To develop a unified approach for the sea surveillance problem for all configurations was a challenge. It is believed that the approach taken achieved a unified approach. The file structure developed is appropriate for all systems, although it is biased for associated memory processing. To show the variation in AM and non-AM, machine code was developed for the GAP and the non-AM solution. Coding for the variant AMs was not performed explicitly. In one instance the instruction complement of the AM was such that GAP coding was a subset of the AM and its effect could be estimated; in a second instance, the GAP was simulated and the variation in time noted.

1.3 PROBLEM SELECTION

The specific problem utilized in the study was selected by RADC. It was recognized that the study results would depend upon the potential that such memories have to aid the problem selected. The general area of the problem was to be data storage and retrieval of formatted files relevant to the intelligence community. The selection criteria utilized to choose the data base and problem were:

(1) Actual Problem. The problem selected should represent an actual problem within the intelligence community.

(2) Problem Solution Exists. A solution to the problem must exist for a conventional computer organization. The solution need not be implemented.

(3) Solution is Operational. There must be at least one operating solution to the problem. (A very strong preference is given to operational solutions.)

(4) Problem Statistics Available. Problem statistics, if required, should either be available or readily accessible to the project. Problem statistics are:

(a) Data base size

(b) Number of files

(c) Record and file relationships

(d) Term relationships

(e) Query complexity

(f) Query volume per day

(g) Maintenance complexity

(h) Maintenance update per day

(i) Query response speed

(j) Maintenance response speed

(5) Representative Problem. The problem should be representative of a class of problems. It should not be special and restricted in form, solution, or need.

(6) Complex Operations. The procedures required in the problem solution should be sufficiently complex to present a challenge in the development of a solution.

(7) Large Data Base. The data base should not be so small as to be trivially accommodated by current equipment and techniques. It must require the best design and equipment for solution.

(8) Potential for Associative Memory. If possible, the problem chosen should have potential application for an associative memory in the design of a solution.

The above criteria were considered as guides rather than absolutes.

The following problems were considered by RADC:

(1) Sea Surveillance. Several data bases concerning ships, their locations, characteristics, cargo, and ports are contained within the sea surveillance problem. Queries and maintenance orders are presented to the computer.

(2) Naval Logistics. Information concerning logistic requirements for ships and ports is maintained. Queries and maintenance orders are of importance.

(3) Army Order of Battle. The order of battle concerning foreign countries is maintained and updated. Peculiarities arise because of the questionable nature of the data received. Processing data into the system and queries is important.

(4) Air Force Intelligence Center. This center maintains, updates, and queries a large data base concerning intelligence information. The data is maintained on tape.

(5) Air Force Command Post. This facility contains inmation (currently stored on disc files) pertaining to airfield facilities, status of forces, war requirements material, strike plans, and strike capability.

(6) Strategic Air Command. Although it is known that a data base exists, no substantial information is known concerning its characteristics.

Figure 1-1 relates the criteria for selection to the above problems. RADC selected the sea surveillance problem since several solutions exist (one of which is implemented) and the types of queries that are posed and the maintenance operations known. This problem is characteristic of a class of problems of interest to intelligence data processing.

In using the sea surveillance problem as our test vehicle it should be clear that no attempt has been made to achieve a formal solution to the entire sea surveillance problem. Where it was believed that the work was not necessary for AM evaluation, portions were omitted or the most convenient method of approach was taken. For example, no attempt was made to define a user's query language since it is known that such languages exist.

	Sea Surveillance	Navy Logistic	Air Force Intelligence Center	Air Force Command Post	Army Order of Battle	Strategic Air Command
1. Actual Problem	X	X	X	X	X	X
2. Solution Exists	X	X	X	X	X	X
3. Solution Operational	X	X	X	X	At one time - no longer exists	X
4. Availability of Statistics	X	could be obtained	could be obtained	could be obtained	could be estimated	could be obtained
5. Representative of a Class of Problems	X	X	expected to be, but not precisely known	expected to be, but not precisely known	X	X
6. Complex Operations	X	probably not	unknown	unknown	X	X
7. Large Data Base	X	X	X	X	X	X
8. Potential for Associative Memory	good	medium	unknown	unknown	good	good

Figure 1-1. Problem Selection Possibilities vs. Selection Criteria

1.4 STUDY RESTRICTIONS

Several study restrictions are important to note as they affect the generality of the results. The restrictions imposed by RADC are as follows:

(1) Small Associative Memories. The study was restricted to include only associative memories within the state of the art. Thus, only small associative memories have been investigated. In particular, the Goodyear Associative Processor was specified by RADC as typical of the technology of associative memories. The observations and conclusions developed therefore do not necessarily extend to large associative memories. The hardware techniques for large AMs will be considerably different from those of the GAP.

(2) CDC 1604-B. The central processor to be utilized as typical was the CDC 1604-B. It should not be construed that this is an ideal central processor. Rather, it was selected because RADC has procured an associative memory to physically attach to the CDC 1604. No attempt was made to determine an "ideal central processor." Features of the 1604-B that proved cumbersome are noted in the appropriate sections.

(3) CDC 818 Disc and Control. RADC selected this disc for the study since it is attached to the CDC 1604-B at the RADC facility. This disc imposes severe restrictions upon the generality of the results. Block sizes of 32 words were found to be too small. The file structure developed to focus-in upon the data was tailored towards the peculiarities of this disc to avoid bias. For a different disc, a modified file structure would have been developed.

1.5 SUMMARY OF VOLUME II CONTENTS

This volume is intended to present technical details concerning the approach. As such, it contains information at a detailed level concerning the hardware investigations, problem analysis, and programming investigations. Relevant observations are contained within each section. The reader would be best advised to read Volume I in its entirety and then refer to the particular section of his interest within Volume II.

Section I of Volume II has set forth the goals and objectives of the study, study methodology and limitations, and rationale for the choice of the specific problem chosen for study.

Section II, a volume in its own right, presents all of the details of the hardware investigations. The reasons for the particular designs chosen for study are presented. For each design developed by AUERBACH, a detailed description of the AM and its interface with the CDC 1604-B is submitted. The programming manual for each AM is also given in this section. The programming manual for the Goodyear Associative Processor (GAP), which was the primary AM to be studied, was given to AUERBACH for use in this study by RADC and is included as Appendix A. The time to execute each instruction in the various AMs are included. Because of the peculiarities associated with GAP, timing of the GAP instructions was developed by statistical means as noted in Appendix B. As requested by RADC, AUERBACH has developed costs associated with producing each device. The information is of sufficient generality so that costing of other memories can be performed provided that they are based on the Goodyear technology as indicated by GAP. Costing figures do not include the research and development efforts that were required to achieve the first device and should be considered as relative costs rather than absolute costs.

As previously noted, the basic problem studied was the sea surveillance problem defined in the Office of Naval Research document.* A description of this problem is contained in Section III. The AUERBACH approach towards developing a system which would store the data files, and respond to queries, new input messages, and maintenance orders is discussed. Although specific queries are listed for the sea surveillance problem in Section III, the approach taken was to permit arbitrary queries to be presented to the system so as not to bias the results for particular queries and thereby have results of general utility. To handle arbitrary queries, an operating system has been developed. The operating system is described in this section. The configuration, which consists of an associative memory, the CDC 1604-B, the CDC 818-B Disc, magnetic tapes, and conventional input and output equipment, is described in Section III.

The data base associated with the sea surveillance problem is too large to be stored entirely within the associative memory. To gain ready access to the data, a disc file is utilized. The particular file structure chosen to focus in rapidly upon the data is a multi-list file structure in which the linkage of records is taken and placed into directories. The file structure was organized to obtain maximum efficiency of the

* Sea Surveillance Data Base Representation as Test Vehicle. Prepared by IBM for the Office of Naval Research, Washington, D. C., Contract NONR 4420(00), June 30, 1964.

CDC 818 Disc. The considerations which led to the choice of the file structure, the precise organization of the files, and their physical location on disc is described in Section IV. A discussion of the implication of associative memories for file structures of the type studied in this report is also presented in this section.

The data base that exists is a dynamic one. New entries arise (e.g., a ship has left port, or a ship is at a new location) which must be placed into the data base. Section V describes the processing that must take place on the input data. As with queries, a generalized approach is taken to the problem. Several significant routines are developed for associative memory processing . These are (1) SPC (selected performance operation) which effectively segments data in the AM and permits operations upon sections of data in AM, and (2) the AM dispatcher which controls segments of data going into AM when the data is larger than one AM in capacity. Both of the routines are used extensively in succeeding sections. It is noted that a tag memory such as the A2 memory (see Paragraph 2.5.4 for a description of the A2 memory) obviates the need for SPO as required by all other AMs (GAP, A1, and A3) studied. The degree of utility of an AM for imput processing is also described.

File maintenance is discussed in Section VI. This subject is related closely to input message processing. During input processing, messages and data are generated which are passed along to the maintenance routine. The data consist of directories that have already been looked up and are appended to the messages by the input message processing function. A data element in a record that must be changed affects not only the data record itself, but all directories that are related. All of this data is stored in an AM. An advantage of AM processing over conventional core processing pertains because a large number of operations is affected in making all of the changes that are required. The A3 memory (see Paragraph 2.5.5), which permits data to be transmitted directly from AM to disc, has an advantage over the other AMs studied, but all AMs show to advantage. Timing of typical situations is included in this section as are other comments concerning AM processing.

A considerable amount of effort was expended on query processing as described in Section VII. The manner in which query generality is achieved is described. Numerous algorithms are developed in the section to effect processing upon the data. The utility of a Tag memory such as A2 is noted. The difficulty with GAP because of its inability to perform data dependent processing without going back and forth between AM and core is eliminated by the A3 memory. (An example of a data dependent operation is a request to get the next highest value based upon a previous search operation. That is, the output of the first search is used as the comparand in a subsequent search.) The processing of Polish pre-fixed strings is performed and comments are made concerning compiler processing. Several routines utilized in the processing are noted as is the timing of typical uses of the routine. Those queries listed in Section III which are amenable to processing are timed for several AM configurations and for a conventional system without an AM. It is noted that if several features are added to the CDC 1604-B, then one of the AMs (A3) is at least as good as the conventional approach while the others are not. More complicated* queries than those described in Section III are required before an AM can be shown to effect a significant improvement over conventional processing.

Section VIII compares the A2 memory to the GAP. It describes how the A2 memory may be programmed to simulate GAP. Thus, the contrast in timing between the two devices can be obtained. It should be noted that A2 is half as costly as GAP and yet the time differential in operations is not that great. Considering the fact that in this type of problem the two memories take the same time to load, the disparity in timing between the two is even smaller. A2 has tag bits while GAP does not. Although GAP is on a special channel to facilitate multi-processing while A2 is not, no advantage could be taken of this feature.

As previously noted, Appendix A reproduces the Goodyear Programming Manual for GAP as received from RADC. Appendix B presents the timing for GAP as developed by AUERBACH. Appendix C describes several routines, presents the coding of these routines both for AM and non-AM configurations. The instructions utilized are tabulated for the reader's interest.

* Requiring more AM processing for each load of data.

SECTION II. HYBRID ASSOCIATIVE CONFIGURATIONS

2.1 INTRODUCTION

This section discusses, from the equipment viewpoint, the hybrid associative configurations that have been studied. Ground rules were established under which the equipment study was conducted. The following paragraphs contain discussions concerning:

(1) The purpcse and philosophy of associative memories.

(2) A description of the particular associative memory implementation employed in this study.

(3) Design parameters and configuration parameters.

(4) A description of the logic of the configurations studied in depth. (The GAP, A1, A2, and A3 configurations.)

(5) A comparison of the configurations.

(6) Some comments on the suitability of the Control Data 1604-B computer as an element of the hybrid configuration.

Throughout this section it is assumed that the reader has some knowledge of the 1604-B computer and some of the previous work done in the field of associative memories.

2.2 GROUND RULES

The ground rules given below have been established by RADC and have set some of the limits of this study. Other limits are implied by these ground rules, either directly or indirectly, or by the amount of effort which was authorized for the investigations. The ground rules which are pertinent to the equipment investigation are:

(1) Any configuration studies must be physically realizable within one year.

(2) The Goodyear Aerospace Corporation associative memory technology is to be taken as representative of technology realizable in one year.

(3) The Control Data 1604-B computer shall be utilized, but may be modified as needed.

(4) The Goodyear Associative Processor (GAP), as defined by Goodyear Aerospace Corporation and RADC, shall be studied.

(5) The Control Data 318 Disc shall be used for storage of the data base in the study.

2.3 ASSOCIATIVE MEMORIES

2.3.1 Purpose and Philosophy of Associative Memories

In the following discussion, the environment of the modern high-speed, large-scale, digital computer is assumed.

The common introduction to digital computers treats the high-speed internal memory of the computer as a set of numbered cubby holes. Each item of information, be it character, word, block, or bit, resides in one of these cubby holes or mail boxes and is identified by the number or address of that cubby hole. Thus, each item stored in the memory of a computer system exists in a unique one-to-one relationship with a number chosen from a dense set of integers. The cubby hole number is commonly referred to as an address of the item and will be so referred to throughout this report.

The names or addresses of each item of information are used in two distinct ways in a computer. First, the address is used in each instruction which requires an operand to specify which of the various items stored in memory is in fact to be the operand of the operation. Second, the addresses are used by the instruction sequencing mechanism of the computer to cause instructions to be executed in proper order and sequence. In fact, it is this direct one-to-one mapping of data to address which is the basis for almost all information transformations and instruction sequencing in a computer. By properly placing his program and data in memory locations, the programmer is free to use all the tricks of indexing and indirect and relative addressing developed by the profession through the years. It is important to realize that in order to use his data and sequence his program, the programmer must know the absolute location in memory or the absolute address of his programs and data, alternatively; address numbers can be bound at execution time by a mechanism supplied either through system software or through special hardware that forms part of the central processing equipment. In any case, some base address must be known for every item of program and data which is to be referenced during a computation. In the end, no matter how remote the programmer is fr n his absolute instruction and data addresses, the computer hardware, when calling for the retrieval from memory of instructions or data, must supply an absolute address and this address must be derived by a mechanism other than the memory itself.

Fortunately, the requirements of an absolute addressing system, though cumbersome, are comfortably accommodated by programming aids and special hardware in a computer. Thus, in general, the programmer need only know the location relative to some reference point in his coding of instructions and data which he references. Indexing mechanisms, indirect addressing, and system software provide for the final binding of relative or symbolic addresses at execution time. *

When a programmer does not know the exact location of the item he desires, it is necessary to search the area of memory which he knows or suspects contains the desired item. Knowing the element size of each item of information, the standard addressing scheme allows a program to inspect each element in the pertinent memory area until the desired item is located. Frequently this process is performed, not to find the item that is identified by such a search, but to find the next item or some item in a fixed addressing relationship to the item found. At times the desired result of such a search is the address of the item found. An associative or content addressed memory is designed to perform this search automatically and in a very short time relative to the programmed search time on a normal computer. The end event is to identify or develop an operand address for a particular operation in the computer. Operand addresses can be had a priori, by calculation, or by search. A priori addresses are supplied at coding time by a programmer who, at the same time, arranges to have his operands placed in the appropriate memory locations. Operand addresses are calculated at program loading time by system software or at execution time by index or indirect address mechanisms. Operand addresses are developed by search during the execution of a program or by the use of an associative or content addressable memory.

The purpose of an associative memory may be viewed as the identification of an address (to be used in a subsequent retrieval) of a desired operand or of some point in known relationship to the desired operand, or as the determination of the existence of a desired

* Amdahl, G.M., et. al., "Architecture of the IBM System/360" IBM Journal of Research & Development, Volume 8 #, April, 1964.

Brooks, F.P., Jr., et. al., "Processing Data in Bits & Pieces" IRE Trans. on Elect. Computers, June, 1959.

Bleanings, G.A., "Indexing & Control-Word Techniques", IBM Journal, July, 1959.

Brooks, F.P., Jr., "Recent Developments in Computer Organization" Advances in Electronics 18 45-64 (1963).

Conway, M.E., "A Multiprocessor System Design", Proc. FJCC Vol. 24, November, 1963.

operand. The ability to perform searches and identify operands in times of the same order or magnitude as normal memory accessing operations appears to have a value in many applications for digital computers. This report in fact is designed to evaluate this value in the processing of large files.

The preceding discussion may be summarized by setting down some desired characteristics of an associative memory. These characteristics are:

(1) The ability to identify those memory locations which satisfy certain criteria.

(2) The ability to determine how many memory locations satisfy a given criteria.

(3) The ability to perform the foregoing operations in times of the order of magnitude of a memory access time of a computer rather than by programmed search.

(4) To employ useful criteria in items (1) and (2) above.

The succeeding paragraphs of this section are organized with the view that an associative memory should identify and allow accessing of those memory words which satisfy given search criteria.

2.3.2 An Implementation of Associative Memory

This paragraph describes enough of the basic Goodyear Aerospace Corporation technology, as embodied in the Goodyear Associative Processor, to provide continuity for this report. Full details on the Goodyear Associative Processor and the technology on which it is based can be found in the Goodyear documentation. *

In the following discussions each major logical element of an associative memory is identified so that the configurations studied may be understood.

* Goodyear Aerospace Corporation, Proposed 2048 - Word Associative Memory Equipment, RADC PR 64-844, GAP-2549, 8 June 1964.

Goodyear Aerospace Corporation, Preliminary Programming Manual for the RADC 2048 Word Memory, Appendix A of Goodyear Aerospace Corporation #AP-112286. "Descriptions of Additions to Contract AF30(602)-3549", n. 2.

The basic elements of a magnetic core memory (whether coincident current or linear select) are shown in Figure 2-1. This part of the basic memory is an array of ferrite devices. It is convenient to consider a square matrix of devices m x n. This matrix may be further organized into n words of n bits each. The memory address register (MAR), which is a register of log base 2n bits, serves to select which of the n memory words will be read or written in access operation. The memory buffer register (MBR), which is a register of n bits, is used to receive the information read from memory and to hold the information to be written into memory.

Two other logical elements are needed for a basic memory, the control and timing section and the interface section. Control and timing logic satisfies the detailed physical and logical needs of the particular memory implementation employed. The interface logic couples the memory system to using equipment. The operating sequences for such a basic simple memory are well-known and will not be belabored here. Such a memory array forms a basis on which an associative memory can be constructed using the Goodyear Aerospace Corporation's technology.

The logical elements which are added to a memory system to provide an associative memory are shown in Figure 2-2. These elements are :

(1) The associative control which provides the necessary sequencing, decision making, and control for implementation of the search algorithms and interfacing functions of associative memory.

(2) A comparand register which will hold the argument for a search.

(3) A mask register which indicates which positions of the comparand register are to be considered during a search.

(4) A logic network which contains one stage for each word to be searched in parallel; this network directly implements the greatest part of the search algorithms and contains identical sections for each word.

(5) The S register or response vector register which will contain, at the end of the search, the result of that search. This register has one bit for each word which may be searched in parallel.

(6) The resolver which resolves from the S or response vector register the address number of the responding words in memory. The number of bits of storage involved in each of the elements in Figure 2-2 is noted on the elements.

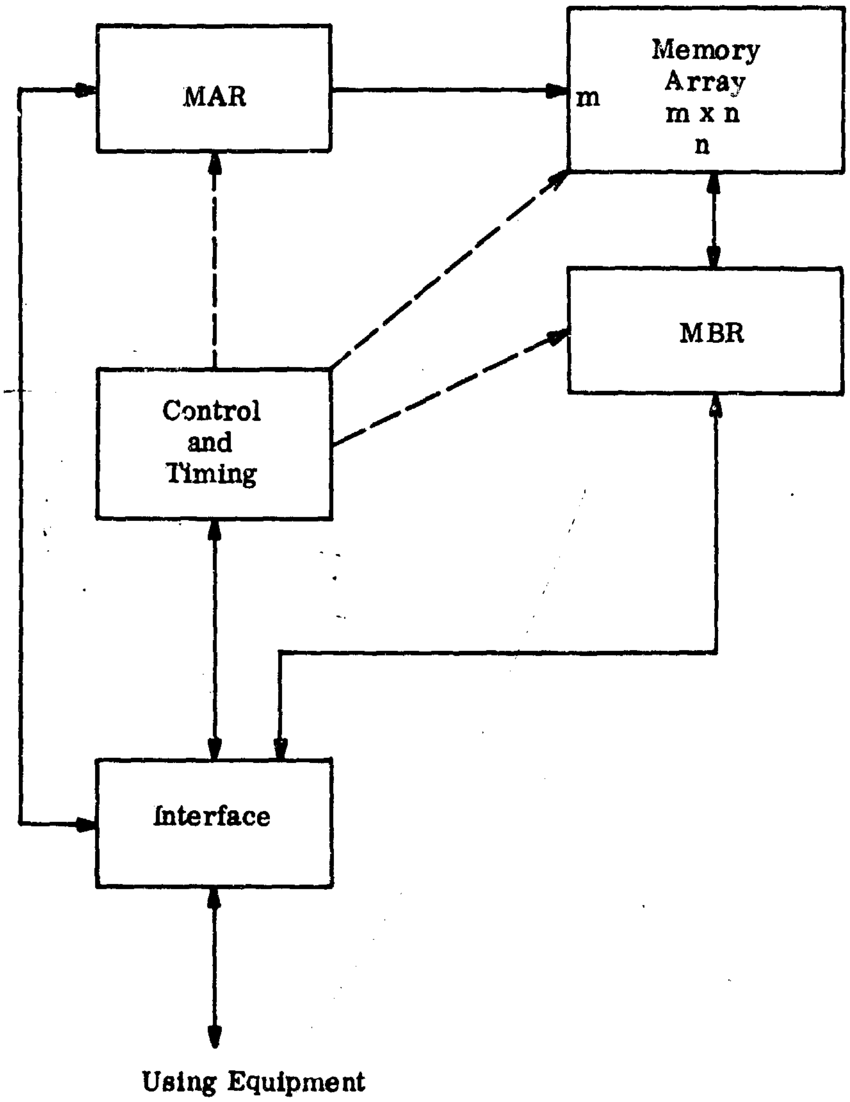


Figure 2-1. Basic Elements of a Memory in Words of n Bits

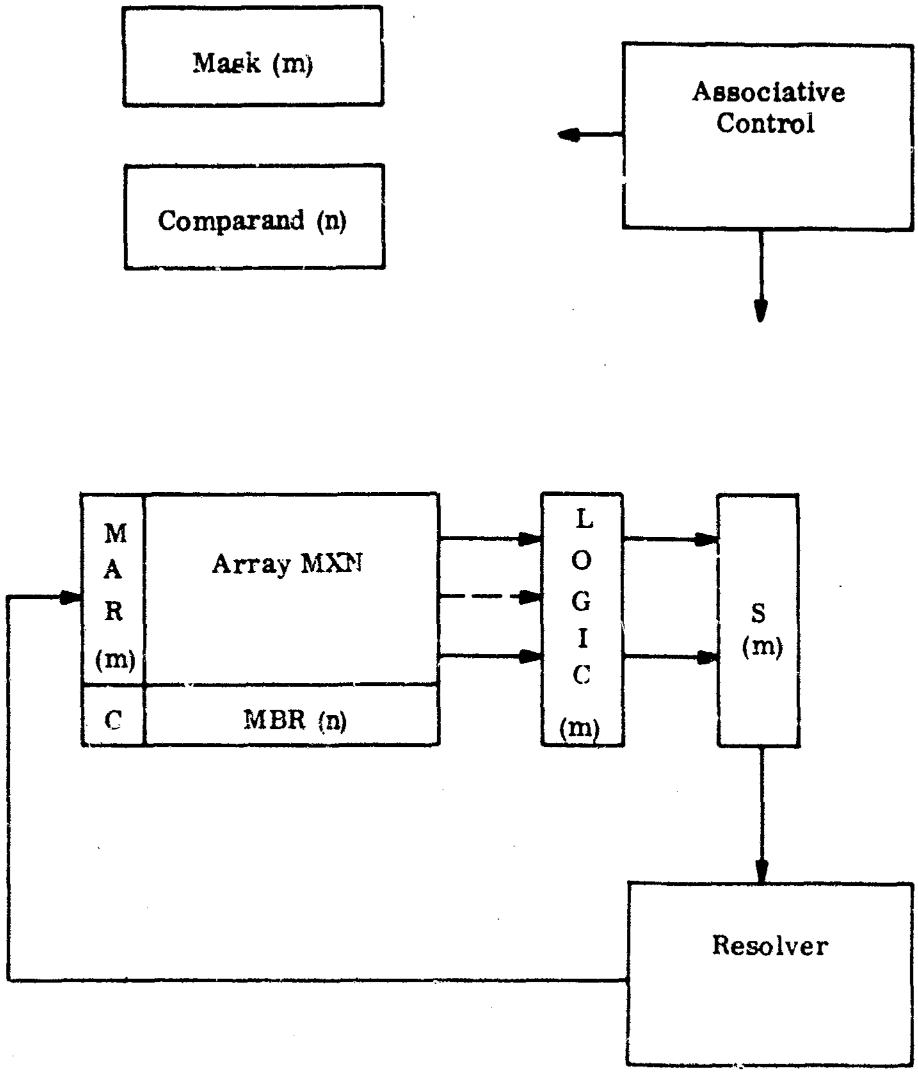


Figure 2-3. Associative Elements Added to a Basic Memory

The logical elements shown in Figure 2-2 satisfy the requirements established in the previous paragraphs. Using the algorithms, given in the Goodyear documentation and others listed in subsequent paragraphs of this report, provides an associative capability. The response vector or S register at the end of the search indicates which of the words searched satisfy the search criteria. The resolver serves to translate this vector notation into address numbers for accessing of memory. Thus, through the response vector and the resolver, all memory words satisfying the search criteria may be read or written on the basis of the search.

In the Goodyear technology, searches are implemented by a bit-serial word parallel scan of the memory array. The most important facet of the Goodyear technology is the speed with which this scan may be accomplished. A column of bits may be interrogated in one-tenth of a microsecond. Thus, a 50-bit word may be searched in five microseconds using one interrogation per bit per search. The implementation of complicated search algorithms frequently requires more than one interrogation per column or it may require supplementary operations between each column interrogation. These times, therefore, are greater than the five-microsecond basic search time of the Goodyear technology.

To implement a 2,000 word associative memory with the Goodyear Aerospace Corporation technology, it is necessary to physically treat each 1,000 words of memory separately. This is indicated in Figure 2-3, which shows the basics of the Goodyear Associative Processor implementation. As shown in Figure 2-3, two half-arrays are employed, each of 1,000 words of 50-bits each. The 50 bits are used as follows: 48 bits for 1604-B data, one bit for parity, and one bit as a "busy bit." One thousand stages of search logic and 1,000 stages of response vector or S register are employed. Half a memory is searched at a time developing a 1,000-bit response vector. To search the second half of memory it is necessary to save the 1,000-bit response vector previously developed. The D and E registers of 1,000 bits each are provided for this purpose. Using this organization the Goodyear Associative Processor may perform a simple search of 2,000 words in 10 microseconds. The column-by-column interrogation of the GAP enables the contents of a given memory column (in this case 1,024 bits) to be read out to the logic circuits. This readout is not destructive. At the logic circuits, the contents of the memory bits may be compared with the search requirements as specified by the search command, the comparand register, and the mask register. As logic is performed on the available information, a response is developed and stored in the response vector or S register. Information from the S register may be transferred to and from the D and E registers in the GAP implementation. The transfers may have certain logical connectors applied with the previous contents of these registers.

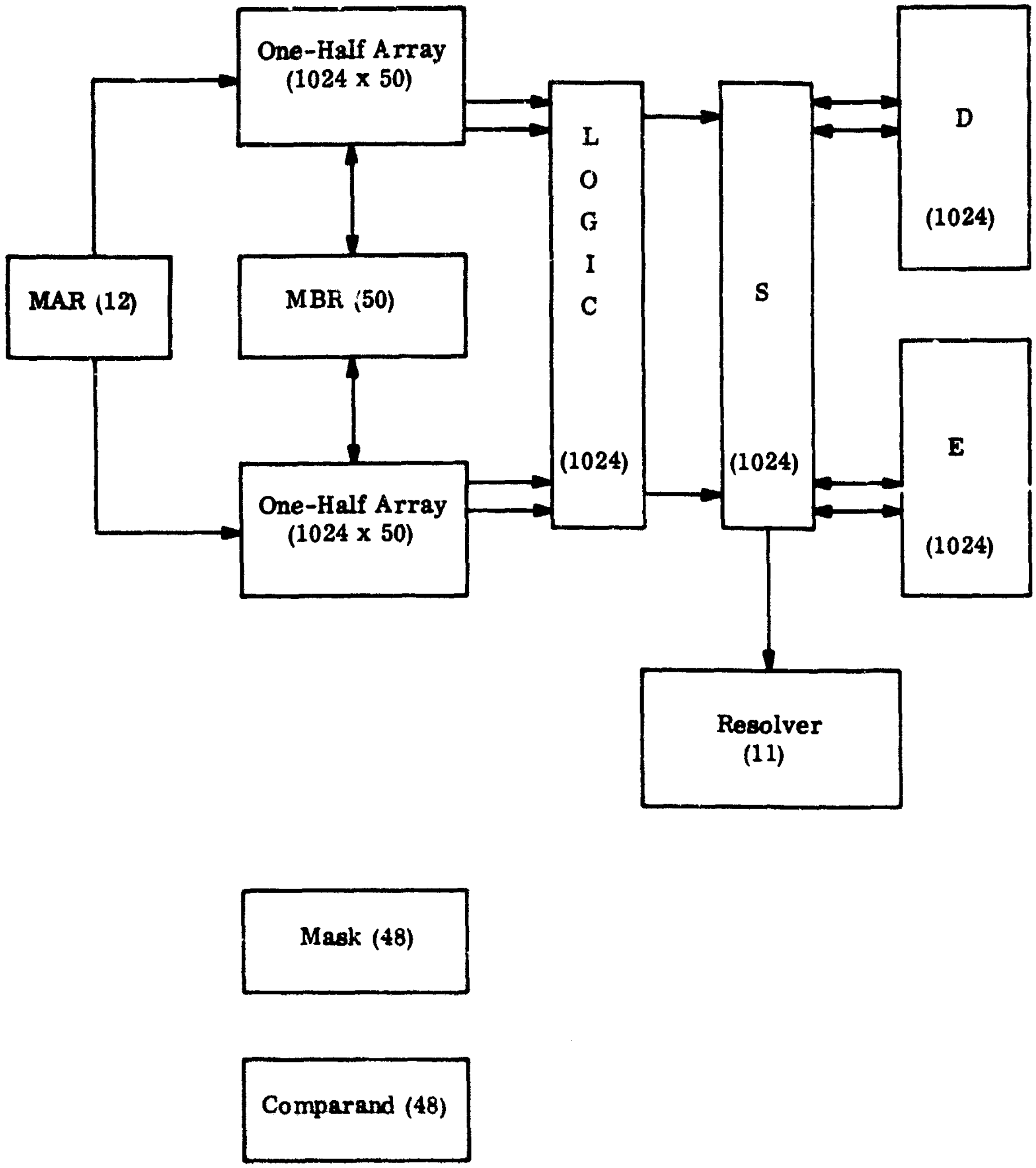


Figure 2-3. GAP Associative Implementation

The important points to note are that search algorithms are implemented by a serial-by-bit scan of the memory words, the interrogation of a memory column is nondestructive and requires one tenth of a microsecond, 1,000 words may be interrogated in parallel, and that it is, in RADC's estimation, economically feasible to provide 1,000 stages of search logic and up to 3,000 stages of response storage with associated logical gating.

Since the primary purpose of this study has been the evaluation of existing technology, not the design and development of associative memory technologies, the Goodyear Aerospace Corporation techniques and, in general, their search algorithms have been accepted. Where different algorithms or techniques seemed beneficial and could clearly be implemented with the existing technology, they have been adopted.

Figure 2-4 summarizes the important elements of Goodyear Aerospace Corporation technology which contribute to the associative memory capability employed throughout this report.

2.4. HYBRID CONFIGURATION PARAMETERS AND POSSIBILITIES

This paragraph discusses the possible variations in logical organization and system configuration made possible by the basic Goodyear Aerospace Corporation associative memory technology. The logical organization of the associative memory and the system configuration parameters are discussed in separate paragraphs. A summary paragraph illustrates the possible system designs that could be achieved using the Goodyear technology and the Control Data 1604-B computer.

2.4.1 Associative Memory Parameters

Figure 2-2 shows the associative elements added to a basic memory to yield an associative memory. Of the elements added, the mask register and comparand register require relatively little hardware. The bulk of associative memory capability is achieved through the very large collections of logic elements associated with the search logic, S or response vector register, and the response resolver and associated control. So far as AUERBACH has been able to determine, the response resolver is already an efficient design for its required task. Since this task must be performed in any associative memory, there is little to be gained from changing the resolver. The remaining blocks, search logic, response vector, and associative control, offer a fruitful area in which changes may be investigated. The search logic and response vector's storage registers in the Goodyear Associative Processor count for approximately 45 percent of the production cost of the GAP (see Paragraph 2.7.).

- NON-DESTRUCTIVE INTERROGATION OF 1024 BITS
- 100 NANOSECOND INTERROGATION TIME
- INEXPENSIVE LOGIC (1024 SEARCH LOGIC STAGES, 3072 REGISTER STAGES, AND GATING ARE ECONOMICAL FOR 2048 WORDS OF MEMORY)
- FAST RESPONSE RESOLUTION THROUGH LARGE LOGIC NETWORK
- SUPERIMPOSED NORMAL MEMORY ACCESSING FUNCTIONS

Figure 2-4. Elements of Associative Technology

It is clear that large reductions can be made in the amount of hardware associated with the response store. Since it was one of the objectives of this study to evaluate the sophisticated associative memory features, cost reductions have been made in the configurations studied.

Figure 2-5 shows the parameters of variance in a study of associative memory logic. The items marked with an asterisk are those which have a potential for relatively great dollar savings in design. Each of these parameters is discussed in the following paragraphs.

2.4.1.1 Number of Different Searches. The searches for which algorithms have been specified by Goodyear Aerospace Corporation are shown in Figure 2-6. Each search listed has been implemented in the Goodyear Associative Processor and accounts for some portion of the search logic in the GAP. However, all searches are not logically necessary; for example, the NOT EQUAL search can be obtained by complementing the response vector of an EQUAL search; the GREATER THAN EQUAL and LESS THAN EQUAL searches are combinations of the appropriate magnitude search with an EQUAL search; the BETWEEN LIMITS search is a combination of the magnitude searches; and the NEXT GREATER and NEXT LESS searches are combinations of the maximum-minimum and magnitude searches. If these functions were not provided, the programmer would have to arrange a sequence of search operations and arrange for the logical processing of the resulting response vectors to attain the same result. The inclusion of a full set of general-purpose searches certainly enhances the capability of an associative memory; however, their inclusion also increases the cost. Thus, the value of a given search category should be determined from an analysis of applications, and the number and kinds of searches included in an associative memory are a parameter of this study.

2.4.1.2 Number of Search Instructions. For a given set of searches implemented in associative memory, the number of possible ways that a using programmer can specify a search is a parameter of the design. The set-up procedures required for institution of a search are also involved here. For example, searches may automatically imply the reloading of comparand and mask registers, or they may imply the shifting of comparand and/or mask registers, or a search instruction may simply start a search algorithm with the responsibility for the proper setting of all auxiliary registers and functions being left to other program instructions. Thus, the number of search instructions that are implemented in a given design will affect the convenience of use; and, depending upon the application, the efficacy of the design and the number of search instructions are parameters of associative memory design.

- NUMBER OF DIFFERENT SEARCHES
- NUMBER OF SEARCH INSTRUCTIONS
- AMOUNT OF RESPONSE STORAGE AND LOGIC
- AMOUNT OF DIRECT PARALLEL HARDWARE
- NUMBER OF AUXILLARY AND HOUSEKEEPING INSTRUCTIONS

Figure 2-5. Associative Memory Parameters

Symbol	Description
$=$	EQUAL
$\neq$	NOT EQUAL
$>$	GREATER
$\geq$	GREATER THAN EQUAL
$<$	LESS
$\leq$	LESS THAN EQUAL
$< y <$	BETWEEN LIMITS
MAX	MAXIMUM
MIN	MINIMUM
MIN	NEXT GREATER
MAX	NEXT LESS

Figure 2-6 Possible Searches

2.4.1.3 Amount of Response Storage and Logic. This parameter involves the number of bits of response storage provided for each word in the associative memory. Due to the technology employed by Goodyear, temporary storage required for shifting and gating functions in the response store must be included here. Because of the physical organization of the Goodyear associative memory, it is possible to have as little as one-half bit of storage per word, whereas the GAP employs well over three bits per word. Associated with the simple storage of response vectors is the logical manipulation. There are 16 logical combinations of two binary vectors, and of these 16, some are trivial, some are almost mandatory, and some are mere frosting. The usefulness of any particular logical function which may be performed between two response vectors is something that must be determined by an analysis of the application or application classes. It is important to remember here that only half response vectors are considered because of the splitting of the 2,000 word memory into two 1,000-word blocks.

2.4.1.4 Amount of Direct Parallel Hardware. The most general and sensible implementation of the response store and its logic is directly in fully parallel hardware. Such parallelism involves from 1,000 to 3,000 stages of highly complex logic. It is also possible to arrange for the use of extremely high-speed, thin-film (or other) memories to provide the storage for response vectors. This extremely high-speed logic is readily available in today's technology. As an example, several thousand bits of response store could be implemented in many hundreds of words of high-speed memory, each word containing on the order of 100 bits. Such an arrangement would imply that the processing of response vectors was performed 100 or so bits at a time requiring considerably less logic than the full parallel processing. The value of the logic saved must be weighed against the possible decrease in effective speed for particular applications.

2.4.1.5 Number of Auxiliary and Housekeeping Instructions. The designer is free within the specification limits of the 1604-B computer to provide many or few convenience and housekeeping instructions for the use of the programmer in manipulating associative memory capabilities An effective set of auxiliary and housekeeping instructions will raise the relative efficiency for given applications by requiring less instructions to be written by the using programmer.

2.4.1.6 Summary. By suitably choosing values for the parameters discussed in the preceding paragraphs, AUERBACH has identified four classes of possible associative memory logic mechanization possibilities. Figure 2-7 shows these classes and indicates that they are

PREVIOUS PAGE WAS BLANK, THEREFORE WAS NOT FILMED

pertinent to AM characteristics. Because the parameters discussed above are not completely independent, it is necessary when choosing the values for the parameters to consider the overall mechanization of the associative memory. Thus, the characteristics chosen for each of the associative memory design parameters in each class reflect an attempt to produce a balanced associative memory design. As shown in Figure 2-7, the four classes of associative memory logic mechanization possibilities are:

(1) Minimum hardware
(2) Minimum parallel hardware
(3) Full hardware implementation
(4) Response store and film memory

The minimum hardware mechanization is chosen to provide a minimum of associative memory capability; other parameters are chosen to be in balance with this basic goal. Mechanization provides minumum convenience, power, and flexibility for the programmer, but still a basic minimum of associative memory capabilities. The full hardware implementation is included first because of the ground rules given previously, and second to determine the effect of the greater speed and flexibility provided. The film memory mechanization is designed to test the power of extremely large response stores with the ability to specify complicated processing of response vectors.

2.4.2 Connection and Configuration Parameters

There are basically four configurations for connecting an associative memory to the 1604-B computer. These are:

(1) A peripheral device.
(2) A multiprocessor.
(3) Integration into the structure of the 1604-B.
(4) A special controller or device.

As a peripheral device, the associative memory may be connected to the buffer channel, the transfer channel, or to a special direct memory access channel as has been done with the GAP. All four classes of associative memory logic mechanization are feasible with a peripheral device connection.

As a multiprocessor, the associative memory may be made to function more or less autonomously with the 1604-B. In true multiprocessor configurations the AM would have access to 1604-E memory as well as the 1604 having access to associative memory.

All AM logic mechanizations except the minimum hardware mechanization are reasonable in multiprocessor configurations.

In integrated configurations the associative memory becomes part of the 1604-B memory. Of course, the part of 1604-B memory that is associative memory has full associative capabilities. Again, all AM logic mechanizations are reasonable except the minimum hardware mechanization.

Special configurations are also possible in which the associative memory becomes an integral part of a disc file controller or a peripheral search device. The efficacy of such configurations is greatly dependent upon how well the detailed design matches the characteristics of the application.

Figure 2-8 displays the possibilities for connection and configurations of the associative memory 1604-B computer.

The associative memory seems attractive as a special input-output controller and search device. This paragraph will set down the basic ideas for these uses. In the processing of large files the major problem is the efficiency of the mass storage media employed to hold the file data. The various mechanical and electromechanical storage media provided by equipment designers go a long way toward providing ready access to any physically located segment of data. However, in most random file processing problems, the critical function is determining where in the physical file the desired data is stored. Various indexing schemes and search algorithms have been developed over the years to aid in rapidly locating a given segment of data. Since the basic problem is the identification of what data is stored where on the physical storage medium, it seems that an associative memory should be able to provide improved efficiency in the use of mass storage devices.

AUERBACH envisions using an associative memory as an integral part of a mass storage control unit. In the case of disc file storage or other locating storage

- PERIPHERAL DEVICE
    - Buffer Channel
    - Transfer Channel
    - Special Channel
- MULTIPROCESSOR
- INTEGRATED
- AS SPECIAL ID CONTROLLER AND SEARCH DEVICF

Figure 2-8. Connection and Configuration Parameters

media, descriptors and continuation instructions received from the file could be processed against search criteria contained in the associative memory to minimize the latency required in the chaining of accesses. In effect, such an operation reverses the role of the associative memory from that normally considered. This means that the associative memory would hold search criteria rather than data to be searched. As material is accessed from the disc file, decisions could be made in real time on the desirability of further processing of material coming from the file. The design of such a configuration would be highly dependent upon the particular file organization developed for a particular application. It is well within the realm of modern computer technology to assemble special-purpose equipment of this kind as needed. Examples of such operations would include the Harvest System and the National Security Agency and the Fixed Plus Variable Structure Computer developed by G. Estrin.*

2.4.3 Summary

Figure 2-9 illustrates in matrix form the AM mechanization possibilities in which complete configurations of an associative memory and 1604-B computer can be assembled. Indicated in the figure by the practical (P), not practical (NP) notation, are those final configurations which, in AUERBACH's judgment, are reasonably balanced and worthwhile systems. Also indicated are the system designations of the configurations which have been studied in depth. These configurations are the Goodyear Associative Processor, the A1 configuration, the A2 configuration, and the A3 configuration. Indicated by an asterisk in Figure 2-9 are those configurations which are considered to be of special promise and interest but have not been investigated fully. The configurations that have been covered in depth were chosen to be, first, representative of the possibilities available, and, second, achievable in the time available for the study. AUERBACH Corporation strongly recommends that RADC pursue further studies in the areas indicated by asterisks in Figure 2-9.

2.5 DESCRIPTION AND LOGIC OF THE CONFIGURATIONS STUDIED

2.5.1 Introduction

This paragraph describes the equipment configurations which have been studied in detail for this report. Emphasis is placed on the associative memory and its integration into the 1604-B equipment complex.

* Estrin, G., "Organization of Computer Systems - The Fixed Plus Variable Structure Computer." Proc. WJCC, May, 1960.

Connection and Configuration Possibilities		Minimum Hardware	Minimum Parallel Hardware	Full Parallel Hardware	Response Store in Thin Film
Peripheral Device	Buffer Channel	P	NP	NP	NP
	Transfer Channel	P (A1)	P (A2)	P	P
	Special IMA Channel	NP	P	P (GAP)	P
Multiprocessor		NP	NP	P*	P*
Integrated		NP	P	P (A3)	P*
Special I/O or Search Controller		NP	NP	P*	P*

<u>Key</u>:

NP: Not a practical system (unbalanced).

P: A practical system worthy of study.

() : The notations in ( ) are the designations of the configuration studied in depth.

* : Indicates a configuration of especial promise and interest.

Figure 2-9. AM Logic Mechanization Possibilities

The basic equipment complex used with the 1604-B is shown in Figure 2-10. The equipment used in this system has been specified by RADC and includes the relatively low-capacity, low speed CDC 818 Disc File System, eight CDC 606 tape transports (operating through two 1607 tape controls) a CDC card reader, a CDC card punch, and a CDC line printer. The channel assignments for this equipment are indicated in Figure 2-10. The configuration of Figure 2-10 is the starting point for the addition of associative memory capability to form a hybrid configuration.

Figure 2-11 shows the equipment organization for the A1 and A2 hybrid configurations. The associative memory system is connected to the 1604-B on the high-speed direct transfer channel (channel 7). This configuration requires no modifications to the CDC 1604-B.

The organization of the Goodyear Associative Processor configuration is shown in Figure 2-12. In this configuration, a special direct memory access and transfer channel (channel 0) is added to the 1604-B; GAP is connected to this channel. This channel contains its own storage registers to hold memory addresses for accessing, does its own counting, and assumes control of the 1604-B during I/O transfer cycles. It is,in effect, an automatic half duplex transfer channel compatibly fitted into the 1604-B I/O structure. GAP contains the logic for most of the required channel control with appropriate control lines made available from the modified 1604-B.

Figure 2-13 shows the overall configuration of the integrated A3 associative memory 1604-B system. The overall block diagram is shown in Figure 2-13(a). As indicated in this figure, the associative memory is an integral part of the 1604-B computer, rather than an external device. Figure 2-13(b) shows the place of associative memory in the internal structure of the 1604-B. As indicated in Figure 2-13(b), the associative memory becomes part of the 1604-B memory. The rest of the 1604-B computer functions normally and, in fact, may use the associative memory as normal memory. Additional control is also provided to implement the associative algorithms of the associative memory. Figure 2-14 shows how the associative memory fits into the detailed memory and logic structure of the 1604-B.

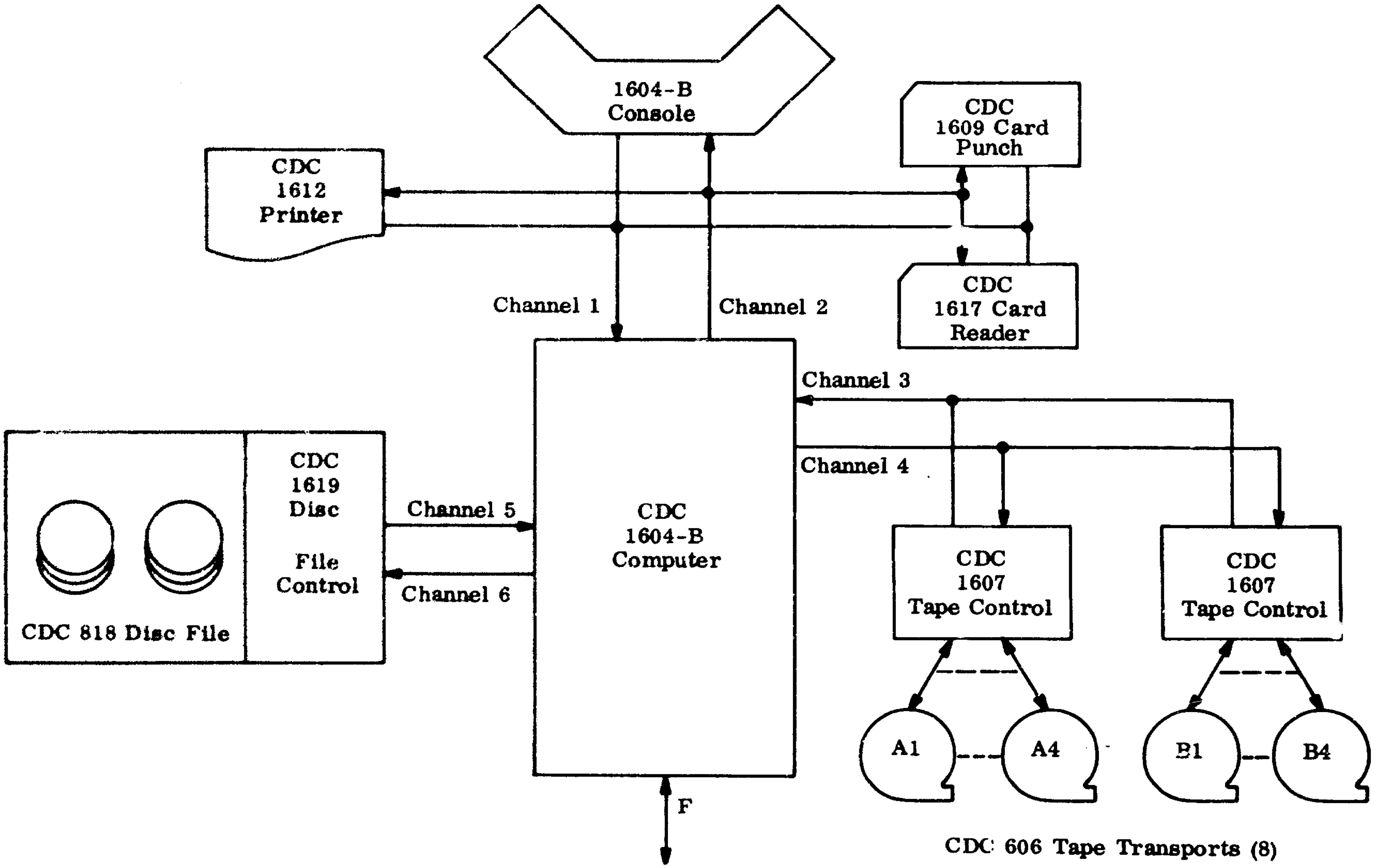


Figure 2-10. Basic Equipment Configuration

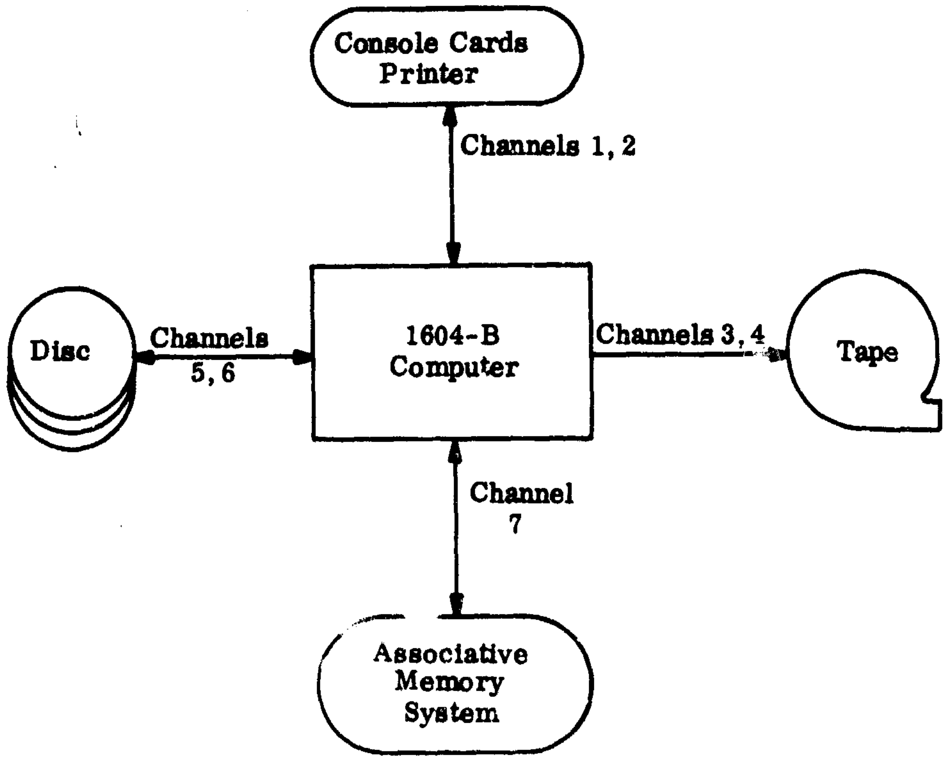


Figure 2-11. Associative Memory as a Peripheral Unit in the A1 and A2 Configurations

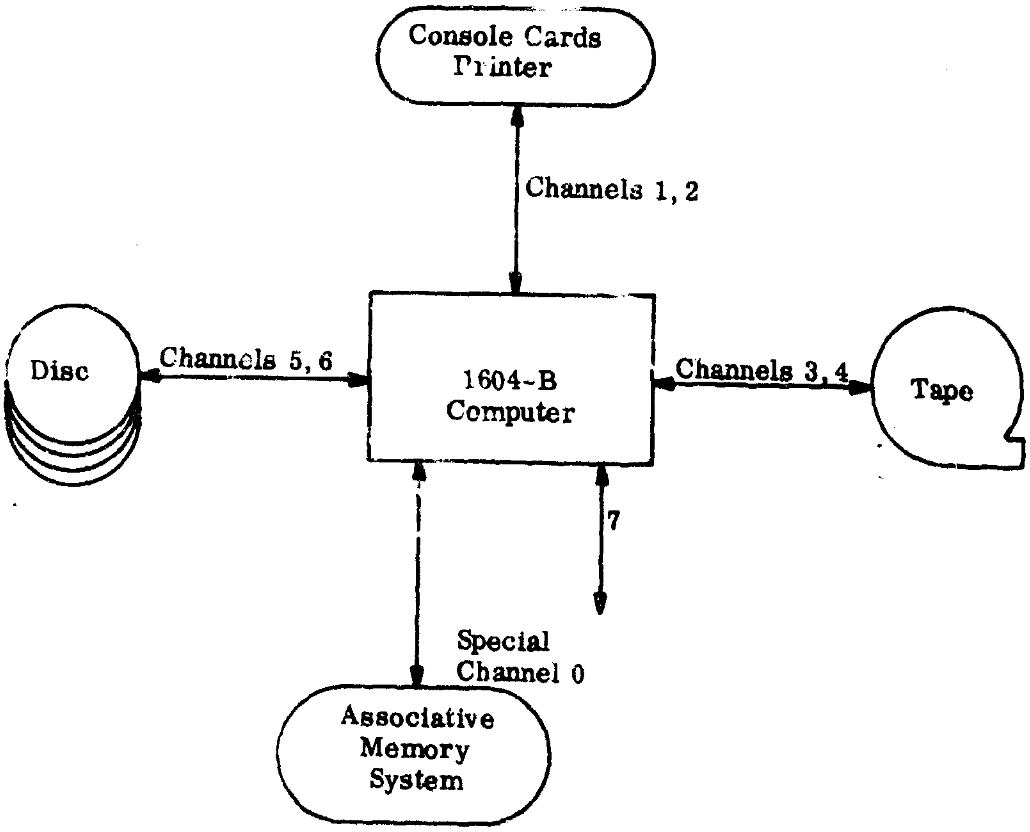
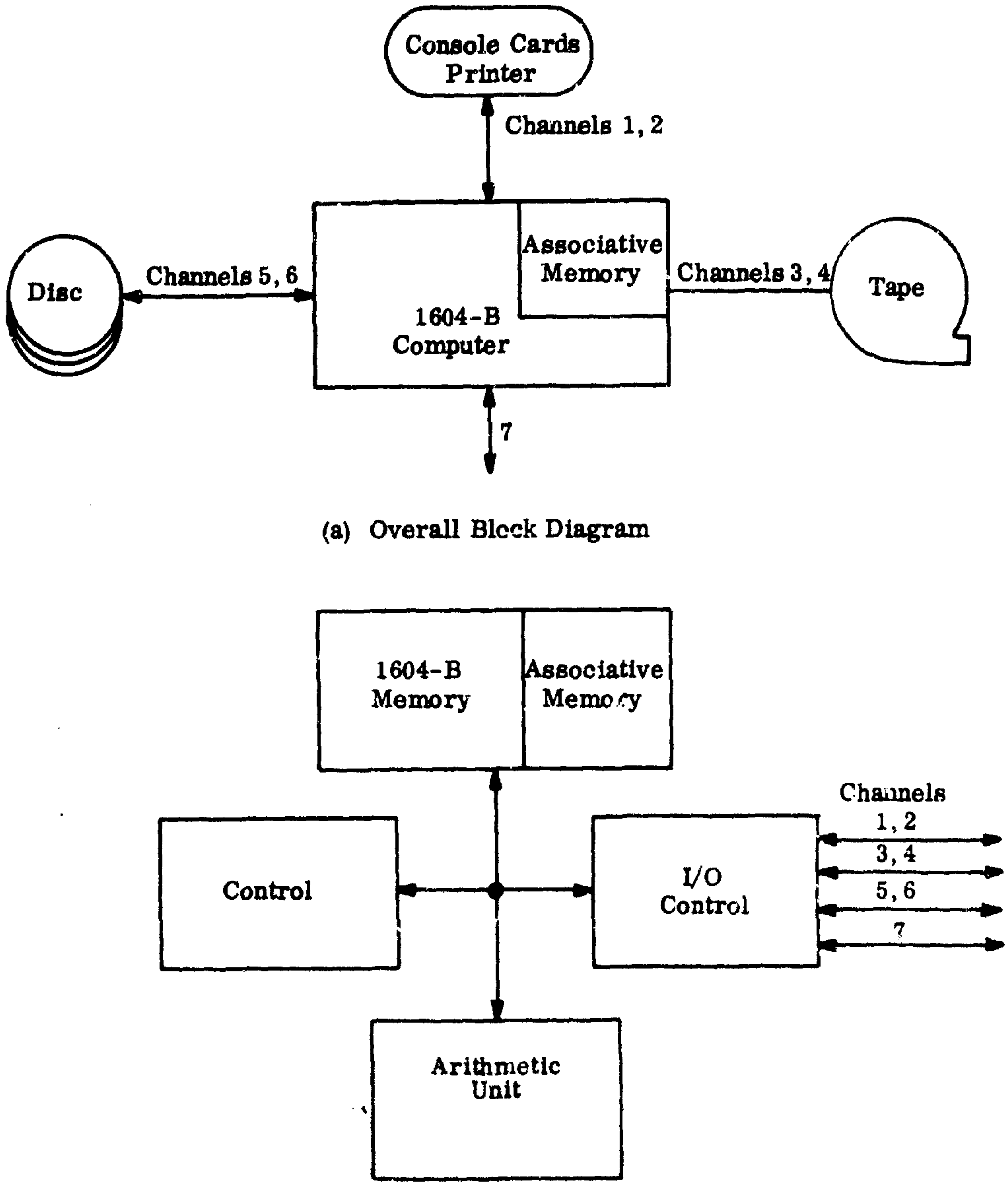


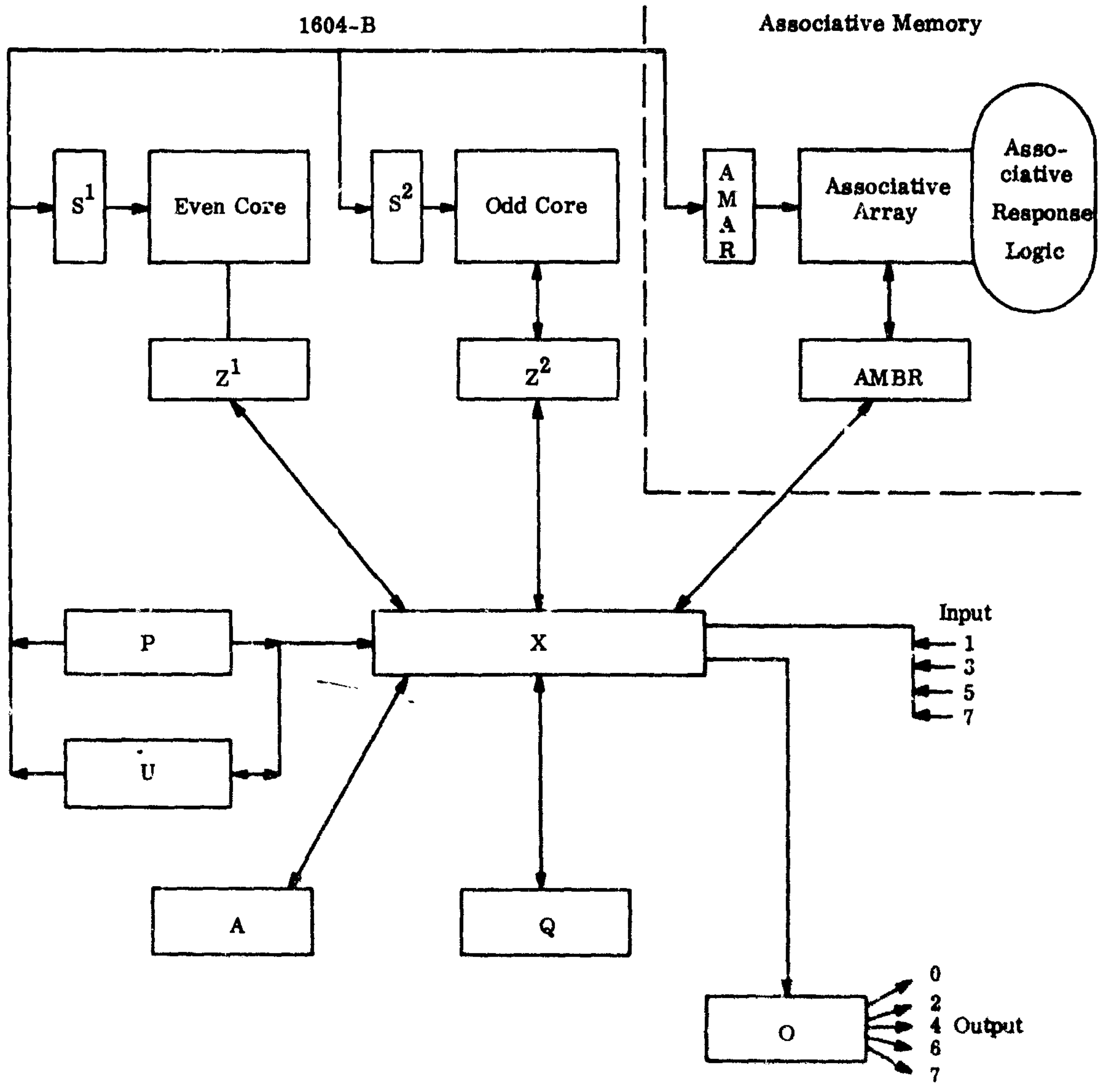
Figure 2-12. Associative Memory in the GAP Peripheral Device Configuration via a Direct Access Special Channel



(a) Overall Block Diagram

(b) Place of Associative Memory in Internal Configuration of 1604-B

Figure 2-13. Associative Memory Integrated into 1604-B as High Order 2048 Words-Controlled by 77 and 00.



NOTE: All registers except AMAR and AMBR are normal 1604-B registers

Figure 2-14. Detail of Integrated Associative Memory Configuration

2.5.2 Goodyear Associative Processor

2.5.2.1 Introduction. The internal organization, construction, and function of the Goodyear Associative Processor (GAP) are described in the Goodyear Aerospace Corporation's proposal to RADC.* The functional use of the GAP is described in Appendix A and in other Goodyear literature.** Appendix A provides all the available information on the functioning of the Goodyear Associative Processor.

In order to accommodate the AM, the following two significant changes have been made in the 1604-B:

(1) A new I/O channel has been added which is similar to the seven existing 1604-B I/O channels and can effectively operate in either a buffer or burst mode.

(2) Through this channel, the AM directly accesses the computer memory, thus eliminating a buffer control word access and related steps in the 1604-B I/O control. The 1604-B location to be accessed is determined by GAP logic, and the address supplied to the 1604-B by GAP.

2.5.2.2 General — GAP Operations. In order to operate GAP, the 1604-B memory must contain the following:

(1) A data area to be operated on by the GAP, if it is to be loaded.

(2) A set of GAP instructions (see Appendix A) that will control AM operation once it is started.

(3) A reserved set of 1604-B memory locations to which are sent GAP responses from the search operations.

(4) The address of the first instruction (item (2) above) in the last 1604-B memory location (77777_8).

Once the above requirements have been satisfied, the 1604-B EXF instructions are used to start the operation of the GAP. Once started, the AM will fetch its

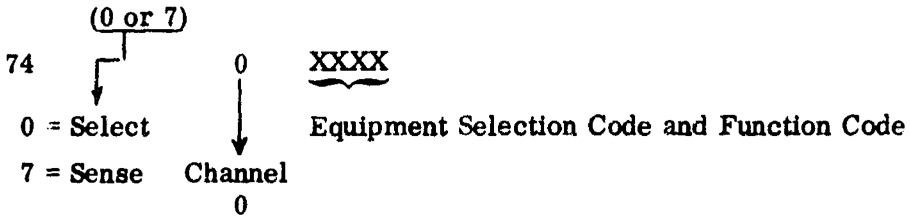
* Goodyear Aerospace Corporation, Proposed 2048 - Word Associative Memory Equipment, RADC PR 64-844, GAP-2549, 8 June 1964.

** "Descriptions of Additions to Contract AF30(602)-3549," n. 2.

instructions and data from the 1604-B memory, operate on the data as instructed, and store the required search responses into the reserved 1604-B memory locations.

2.5.2.3 Direct Access Channel (Channel 0). GAP is connected to a special input and a special output channel that communicate directly with the 1604-B Input-Output section as shown in Figure 2-15. Input and output transmission over these channels will not occur simultaneously. These channels are referred to as the direct access channels since the GAP can specify to the 1604-B the address to be accessed (via 1604-B register C2 and GAP data and instruction address register).

2.5.2.4 1604-B External Function (EXF) Codes. GAP recognizes EXF codes assigned to it as do other peripheral devices. These EXF instructions activate the GAP, sense for specified conditions, and select operations. EXF codes assigned to GAP are in the form (octal) shown below:



(1) Select EXF Codes. The following select codes have been assigned to GAP.

(a) Force. GAP will resume or start operation. The GAP program counter is forced to all ones (77777_8). This is the next instruction address; i.e., this will be the first 1604-B memory location that will be accessed by the GAP. The location, 77777_8, will be accessed for an instruction fetch.

(b) Resume. GAP will resume operation. The next instruction address (i.e., 1604-B address) is located in the GAP program counter prior to this operation.

(c) Clear. GAP will halt (go inactive). All indicators that can be sensed will be reset. All registers will be left in their present state.

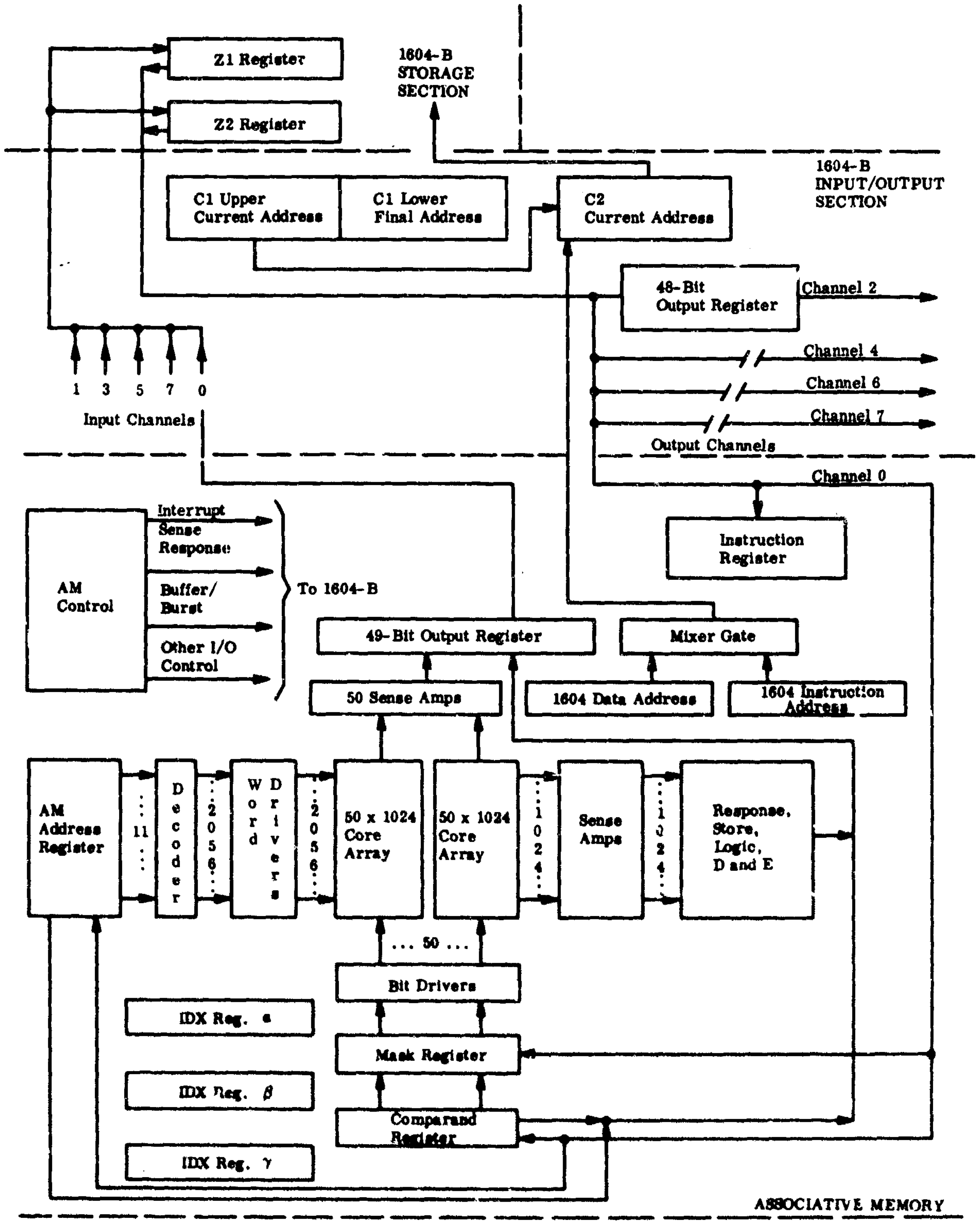


Figure 2-15. Block Diagran ·f GAP Connected to the 1604-B Computer

(2) Sense EXF Codes. The following sense codes have been assigned to GAP.

(a) Sense-Activity. GAP will detect a sense activity operation. If GAP is active (in operation), the sense response line will be raised; if GAP is not active, the sense response line will remain down.

(b) Sense-Parity. GAP will detect a sense parity operation. If there has been a parity error detected since the previous clear operation, the sense response line will be raised; otherwise, the line will remain down. Parity is checked each time GAP location is read.

(c) Sense-Overflow. GAP will detect a sense overflow operation. If any load operation has caused an overflow indication to occur, the sense response line will be raised; otherwise, the line will remain down. Overflow occurs when more words are to be loaded into GAP than there are available locations.

2.5.2.5 Buffer Burst Mode. GAP has the capability of operating in either the buffer or burst mode. To the GAP, the only difference between the burst and buffer modes is that GAP receives data sooner after requesting it in the burst mode than in the buffer mode. GAP will request the 1604-B for the burst mode under block transfer conditions not yet specified by Goodyear, but the 1604-B program will determine whether or not to allow it. (Presumably there is an EXF select code that will inhibit GAP from entering the burst mode, but this has not been specified.) The buffer mode does not stop the main program from running, but the burst mode does, in addition to inhibiting all other I/O operations as well.

When the programmer determines that a burst operation is allowable, he sets a time delay of X* milliseconds (another EXF code must be defined). GAP then has X milliseconds to operate in the burst mode, and at the end of this time data exchange will continue in the buffer mode. If the data transfer is completed in less time, the GAP will take the 1604-B out of the burst mode.

* The limits of the burst time duration are still undefined.

2.5.3 The A1 Configuration

2.5.3.1 Introduction. This paragraph discusses in some detail the design of the smallest associative memory configuration considered in this study. The characteristics of the small AM serve as a basis for initial performance analyses. The detail provided in this paragraph will serve as a basis for the subsequent descriptions of the larger and more complex configurations.

The A1 configuration is described from three viewpoints:

(1) Method of integration with 1604-B computer,

(2) Description of AM including its organization,

(3) The AM instruction set.

The A1 configuration is not the absolute minimum (in terms of hardware) that can be devised, but rather it is as small as seems "reasonable." The "reasonableness" is, of course, subjective but since the goal is to define a small AM for comparison with several others, it is felt to be justified. As an example, the AND or OR connect capability is excluded since it requires an additional temporary storage of one bit per word. If this storage were obtained using flip-flops, the memory cost would be increased by the loaded cost of 20%, if it were obtained using additional bits in the core array, a memory read feature would have to be included which is also costly. On the other hand, three basic search instructions, rather than a single "equality" search, have been included because the performance is significantly improved at a relatively small cost.

In other cases, aspects of A1 organization (i.e., Load Preset, Load One Word, etc.), frills will be added where increased hardware costs are trivial and where programming advantages seem to result. Paragraph 2.5.3.4 indicates which instructions of each feature must be evaluated on the basis of such factors as frequency of use, relative execution time as performed in the 1604-B, and A1 cost of inclusion, etc.

Some supporting detail is also given along with the description of A1 below so that the reader may better understand why some of the choices were made.

Throughout this paragraph Falkoff's notation is used. *

* A. D. Falkoff, "Algorithms for Parallel Search Memories," JACM, Vol. 9 No. 4, pp. 488-511; Oct. 62.

2.5.3.2 Integration with the 1604-B

(1) Connections. A1 is designed to be used on the transfer channel (channel 7) of the 1604-B because none of the AM execution times take more than a few microseconds. There is no advantage in using the buffered I/O channels and, in fact, the programming overhead will be less using the transfer channel. Control of the AM is through the 1604-B EXF instruction. Data transfer is accomplished by INT and OUT instructions.

There are 69 signals between the 1604-B and A1 distributed as follows:

48 for data
3 for A1 and other peripheral equipment identification
9 for control with EXF instructions
9 for synchronizing.

The 48 data signals are used for four purposes:

(a) Loading the associative array (48 bits).

(b) Transferring comparand ($\underline{X}^*$) or mask ($\underline{m}^*$) (48 bits).

(c) Specifying the starting address (a) and number of words (n) for loading the associative array (11 plus 11 bits).

(d) Reading addresses of responders from the AM (11 bits of AM address plus 4 bits from preset register). (See Load Preset instruction.)

The nine control signals or external function signals are used for two purposes:

(a) Transferring A1 instructions to A1.

(b) Specifying sense conditions to the A1.

The purposes of these signals are explained in more detail in Paragraph 2.5.3.4.

The nine synchronizing signals are not of particular importance here except to note that since the AM is to be used on the transfer channel and since data or control signal transfer only takes place in one direction at a time, four of the synchronizing signals provided on the 1604-B buffered channels are not used.

(2) Timing of Transfers. All transfers are completely dependent on the 1604-B timing, and various average times have to be used depending on other I/O operations, location of INT and OUT instructions in upper or lower instruction slots, and 1604-B memory locations referenced.

2.5.3.3 Description of A1

(1) Organization. The organization of A1 is explained with reference to Figure 2-16, which shows the main components of the A1 and the more important data transmission paths.

Instructions sent to A1 by EXF signals are synchronized by the SYNC circuits and interpreted by the control section. The associative array is loaded by transferring 48-bit words into the data circuits, then transferring them to the X register which, along with the M register, controls the data entry into the core array through the bit circuits. The data word is loaded into a specific array location specified by the A register, even though the instruction being executed may be "load the data words into those locations responding to the last search." In this case the A register would have been loaded sequentially from the S register scanner. The connections for this transfer are omitted from Figure 2-16.

During execution of a search instruction, the results of each interrogation are sensed by the sense logic and stored in the S register. To reduce the amount of logic hardware required per word, the array is divided into two 1,024 word parts; only 1,024 sense amplifiers and 1,024 S flip-flops are used. This means that searches can only be carried on in one-half of the memory at a time, the upper or lower addresses. The separation is by the most significant bit of the 11-bit address.

Forty-nine bits are used for data storage. The 49th bit is used by the memory itself as a tag bit (busy-bit) to mark those locations currently storing data. The tag bit is set to a "1" or a "0" when data is stored in that location under program control.

Readout of the A1 is by means of the scanner and output register. Addresses of responders to searches are successively determined by the scanner, and placed into the output register to be read by the 1604-B input channel.

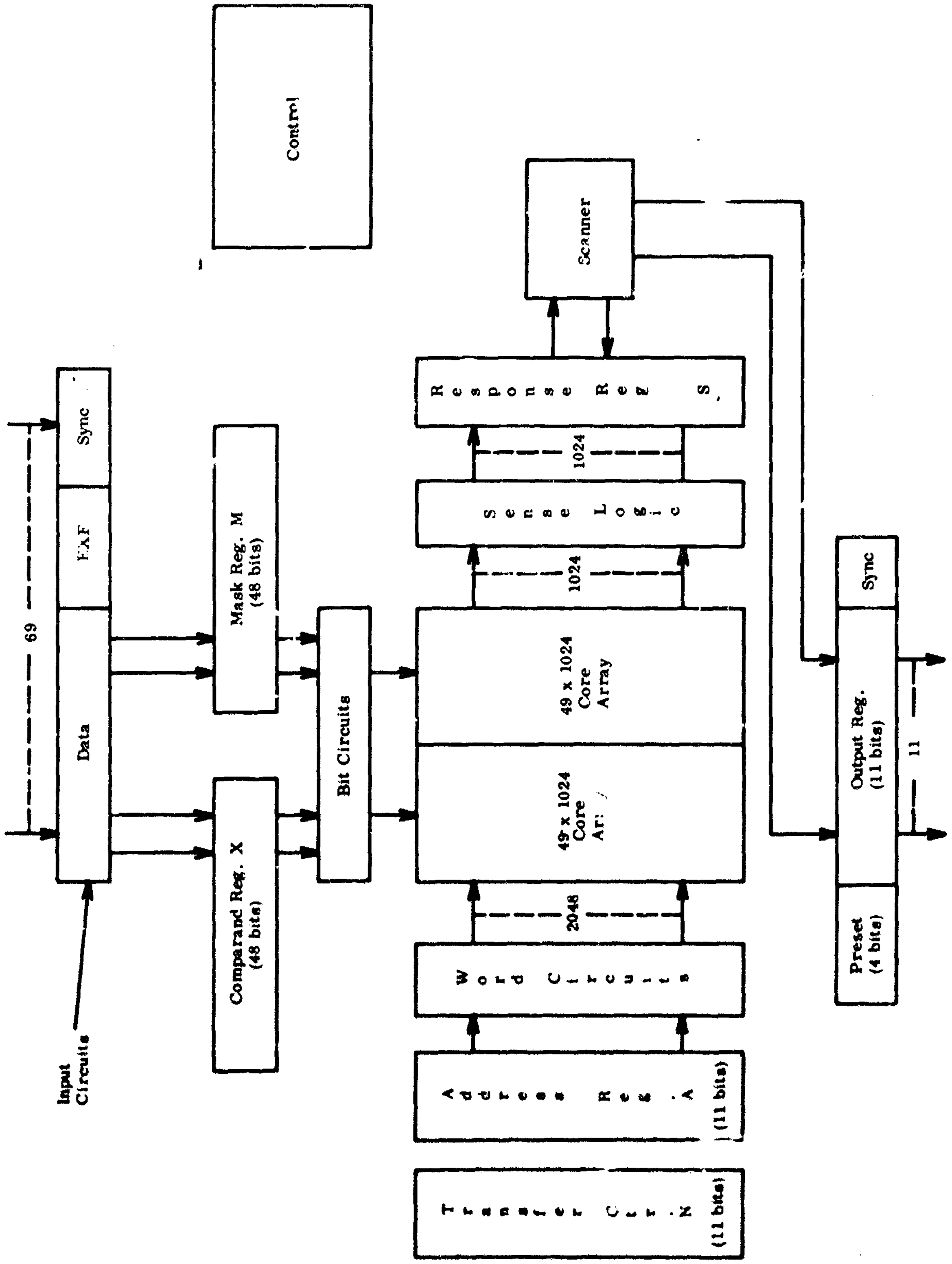


Figure 2-16. Logic Block Diagram for Small AM

The scanner consists of a 10-bit counter and set of gates which can sequentially scan the S register flip-flops (see Paragraph 2.5.6). Its "base" position is the count 1023 which is the address of the "highest" S flip-flop. If an instruction references the upper half of the A1, an 11th bit in the scanner is set so that the scanner appears to be looking at the S flip-flops corresponding to addresses 2047 to 1024. The scanner counter counts downward only.

In the LPS instruction, a "block load on S" instruction, the scanner first looks at address 2047 if higher addresses are referenced, or address 1023 if lower addresses are referenced. If S_{2047}, for example, is in the state required by the instruction, the address 2047 is transferred into the A register and the data word is loaded. If S_{2047} is not in the desired state, the scanner is decremented by 1 and S_{2046} is examined, and so on. When the scanner counter reaches 1024 (or 0000), it stops. If the 1604-B attempts to load another word, the "load overflow" flip-flop will be set and an interrupt generated. The scanner counter may be sensed for the 0000 (1024) state by a sense instruction.

Actual data words cannot be read from the A1.

The preset register is used to specify the most significant four bits of the 15 bits read from A1 into the 1604-B. This feature is included to provide flexible mapping of A1 addresses into 1604-B addresses without the necessity of having to modify each A1 address after it is loaded into the 1604-B.

(2) Response Store. A one-bit cross section of the 1024-bit comparison logic (in Figure 2-16 this was included in the Sense Logic block) and the 1024-bit S register is shown in Figure 2-17. This configuration was selected after comparison with the logical configurations shown in Figure 2-18. The paragraphs below discuss the performance and cost of these four possible configurations and give the reasons for selecting the one shown in Figure 2-17.

The configurations shown in Figure 2-18 provided for eight simple search criteria. The search criteria are:

(1) less than (<)

(2) less than or equal (≤)

(3) equal (=)

(4) unequal (≠)

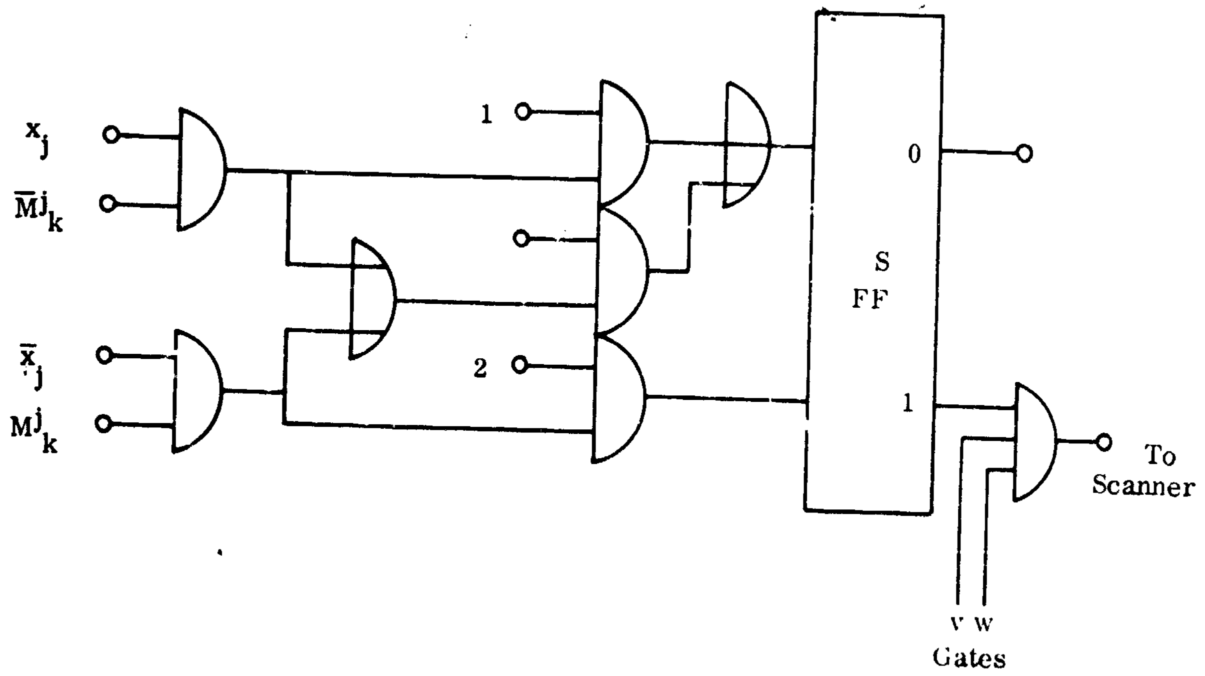


Figure 2-17. Logic Cross Section

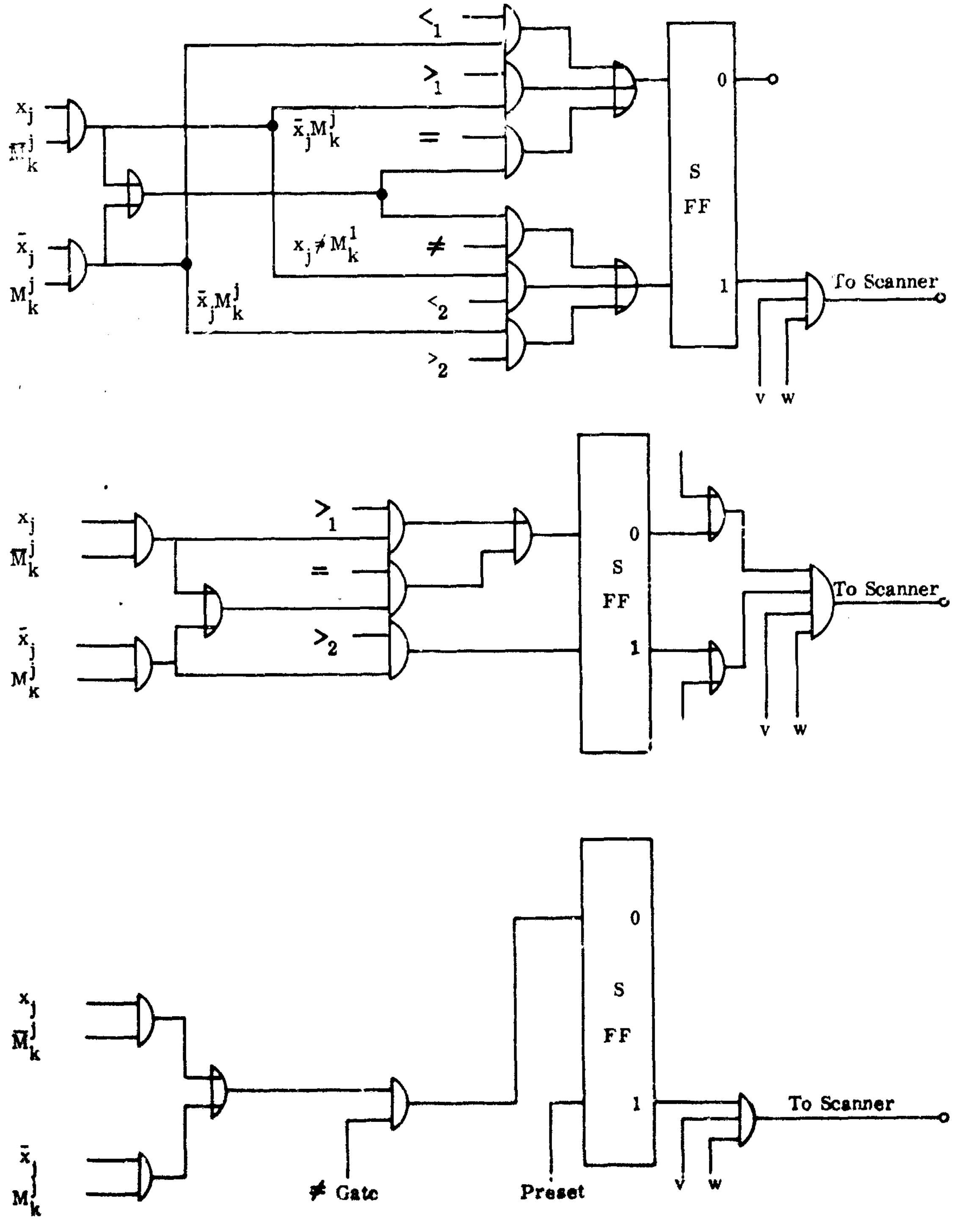


Figure 2-18. Alternate Logic Configurations

(5) greater than or equal (≥)

(6) greater than (>)

(7) equal, and $\underline{S}$

(8) unequal, or S

The first, (a), uses a relatively large number of input gates (8) to sense for various input conditions.

Only four of the criteria are sensed directly by the input gates as shown by the symbols associated with the gates in the figure. These are 1, 3, 4, and 6. Criteria (2) and (5) are obtained by using different initial conditions, then using the "less than" or "greater than" gates. Similarly, (7) and (8) are obtained by using different initial conditions, then using the "equal" or "unequal" gate. This configuration (Figure 2-18(a)) could also be used for other searches (maximum, for example) by additional equipment.

The most obvious way to initialize the S flip-flops is to use preset inputs on each of the two input OR gates. However, these are not used (except in one special case) in the configurations discussed because the presetting function is provided by forcing all X inputs to a "1" and gating the resulting signal into the set or reset side of the S flip-flop as desired. The V and W inputs to the output gate are simply to allow two dimensional addressing of the S register for readout by the scanner.

All of the input gates shown in Figure 2-18(a) are not necessary, since three of the search criteria are complements of the other three. The configuration of Figure 2-18(b) makes use of this fact to eliminate three of the input gates. Complements are formed by selecting the appropriate side of the flip-flop for readout.

One of the output gates of Figure 2-18(b) is unnecessary since complements are not required of each individual S flip-flop but only of the S register as a whole. The S register is to be read (or sensed) serially so that a single pair of gates can be used in the scanner to invert each of the flip-flops as they are scanned. This is done for the configuration shown in Figure 2-17.

Since a low-cost A1 is being defined, the configuration shown in Figure 2-18(c) was also considered. The further reduction in the number of gates as shown is obtained by eliminating the "search for greater than" function.

If the four configurations are compared, the first three are found to provide essentially the same capability — to directly sense for three conditions and allow these conditions or their complements to be sensed by the 1604-B. These can be considered as allowing six search instructions as compared with two instructions obtained from the configuration in Figure 2-18(c). To help decide whether the configuration in Figure 2-17 or the configuration in Figure 2-18(c) would be a better choice for an initial definition, Table 2-1 was constructed. Two measures of complexity (or cost) are shown in Table 2-1:

(1) The number of gates used.

(2) The total number of gate inputs used in each configuration.

Both are included because it is not known exactly how one would choose to build the circuits, and costs might be more accurately estimated by one or the other depending on the technology used. For example, if diode logic were used, the number of gate inputs is more nearly proportional to the number of diodes used and, therefore, is a better measure of cost than the number of gates. On the other hand, if integrated circuits were used in which the number of encapsulations (number of cans), rather than the number of leads, was more important, then the number of gates would provide a better estimate.

Each of these complexity measures is divided by the number of instructions that each gate allows to generate two "figures of merit" —

(1) the number of gates per instruction, and

(2) the number of gate inputs per instruction.

Since both show that the configuration shown in Figure 2-17 has the lowest "cost" per instruction, it was chosen as a basis for A1.

TABLE 2-1. COMPARISON OF RESPONSE STORE CONFIGURATIONS

CONFIG	NO. OF GATES	NO. OF GATE INPUTS	NO. OF INSTRUCTIONS	GATES/ INSTRUCTION	GATE INPU INSTRUCT
Figure 2-18(a)	12	27	8	1.50	3.37
Figure 2-18(b)	10	22	8	1.25	2.75
Figure 2-17	8	17	8	1.00	2.13
-18(c)	5	11	4	2.50	2.75

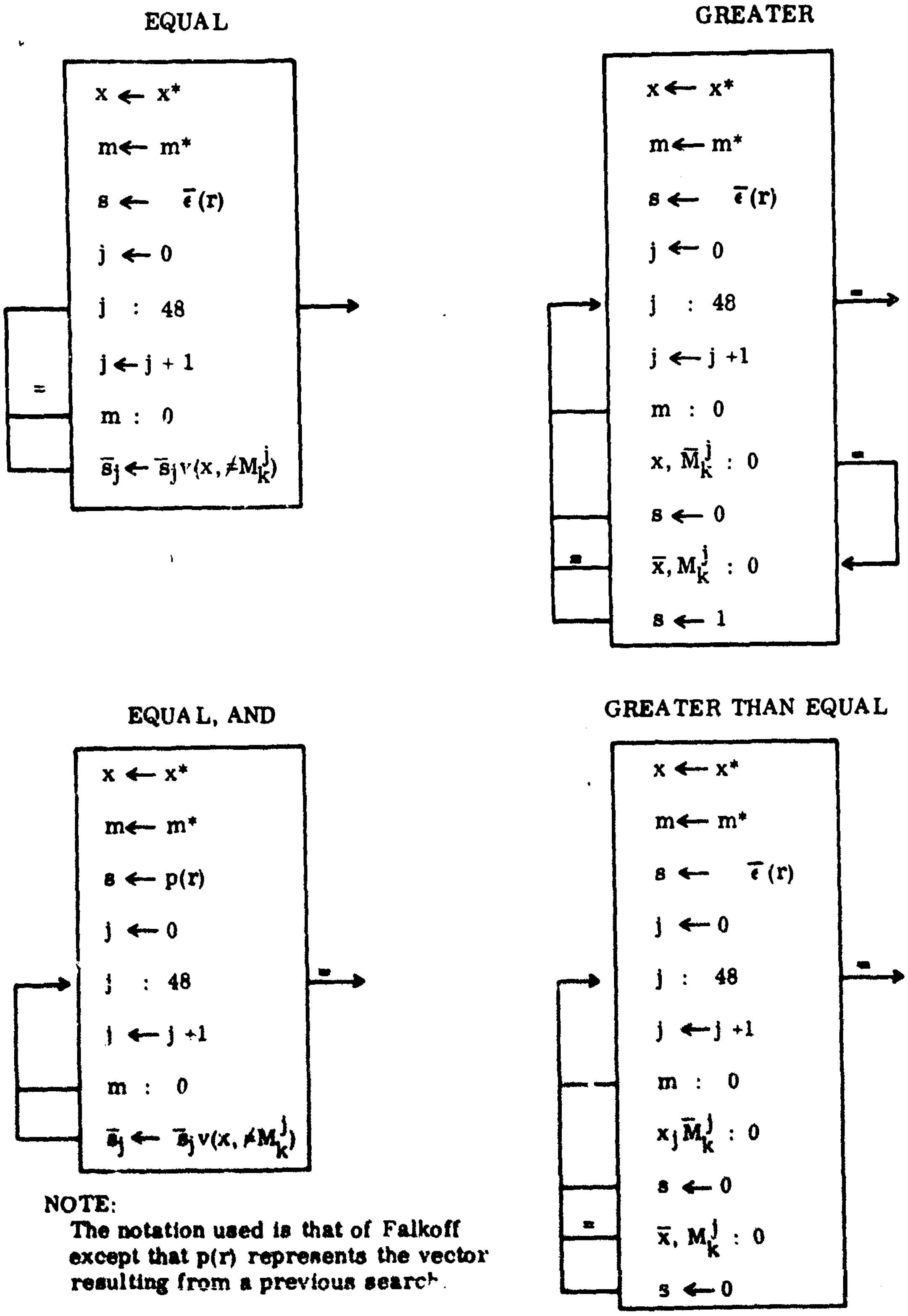
(3) Search Algorithms Used. The algorithms shown in Figure 2-19 indicate generally the approach taken for each search but are not complete in the sense that all hardware actions are included Note that the EQUAL, AND search is the same as the EQUAL search, except that the S register is initialized to the results of the previous search rather than all "1"s. Similarly, the GREATER THAN or EQUAL search is the same as GREATER THAN search except that the S register is initialized to all "1"s rather than all "0"s. The GREATER THAN or EQUAL search, and the EQUAL, AND search, then, cost almost nothing since no per-word hardware is added. The only additions are a couple of control flip-flops and decoding gates.

2.5.3.4 Instruction Set. Instructions have been chosen on the basis of the response store characteristics, anticipated programming techniques, and consistency with other 1604-B I/O devices. This paragraph describes those instructions by first describing the instruction structure and then describing what each instruction does.

(1) Structure. All instructions are sent to A1 by means of the 1604-B EXF Select (74 0 XXXXX) or EXF Sense (74 7 XXXXX) instructions in the same manner as other I/O devices. As can be seen from Table 2-2, the select instruction will be used to give operating instructions and define conditions; and the sense instruction will be used to sense various conditions within A1. The first octal character, X, will be used to specify that channel 7 is to be used, and the second will specify the sub-channel which the AM is using. This leaves three octal characters or nine bits for select instructions and nine bits for sense instructions. Since there are 20 select type AM instructions, five of the nine bits are used for operation codes. Each of the α , β , and γ fields shown in Table 2-2 requires one bit, so a total of eight bits is required, and there is sufficient room in the 1604-B EXF format for the AM instructions. The α field is used to specify whether the S register is to be associated with higher or lower addresses. In other than search instructions, β is used to specify whether "1"s or "0"s are to be sensed in the S register and γ is used to specify whether the tag bit is to be set to "1" or "0".

All but one instruction takes less than eight microseconds to execute; therefore, A1 operations will be finished before the EXF execution is completed within the 1604-B. The exception is the RCS instruction which requires 8 + 0.1n microseconds, where (n) is the number of S entries to be counted.*

* See the discussion of timing of the scanner in Paragraph 2.5.6.



NOTE:
The notation used is that of Falkoff except that p(r) represents the vector resulting from a previous search.

Figure 2-19. Search Algorithms

TABLE 2-2. A1 INSTRUCTION SET

EXF SELECT CODES

Mnemonic	Instruction	Fields
LSL	Load Specific Location	γ
LPS	Load Per s	α β γ
LOS	Load One Word Per s	α β γ
LDA	Load a, n Registers	—
LDX	Load x	—
LDM	Load m	—
LDP	Load Prefix	—
SOE	Search on Equality	α β γ
SAE	Search on Equality, AND	α β γ
SOG	Search on Greater	α β γ
SGE	Search on Greater or Equal	α β γ
RCS	Read Count of Responders	β
RSS	Read Short Count	β
RAS	Read Address of Responders	β
RNS	Read Next Address	β
IOE	Interrupt on Any Error	—
IIE	Inhibit All Interrupts	—
CHC	Channel Clear	—
CAM	Clear AM	—
CIO	Clear Interrupt Only	—

EXF SENSE CODES

TRA	Alive	α
TRY	Ready	α
TIA	Interrupt Active	α
TLO	Load Overflow	α
TUO	Unload Overflow	α
TLU	Load Underflow	α
TUU	Unload Underflow	α
TFS	Scanner at 0000, 1024	α
TES	+/s = 1	α
TLS	+/s < 1	α
TGS	+/s > 1	α

Load instructions are performed through a mask so that only those positions of the comparand, (x), corresponding to "1"s in the mask (m) will be loaded. All other bits will be forced to "0". This unfortunate situation of forcing all masked bits to zero is brought about because the physical arrangement of the memory array requires that a whole word be written at once, and there is no data reading facility to find and hold temporarily the contents of the word being written into.

(2) Description of Instructions

LSL — Load Specific Location, γ

This block load instruction conditions A1 to load one or more 48-bit words from the 1604-B into contiguous locations in the AM array under control of a 1604-B OUT instruction. Thus, the next 1604-B instruction which refers to the transfer channel following the EXF 0 7 LSL γ would have to be OUT b m (1604-B notation). Words will be loaded into numerically decreasing AM addresses until one of three situations occurs:

(a) The transfer is sensed to be complete by the (n) counter. If the 1604-B attempts to continue loading the AM, an error condition exists and the AM overflow flip-flop will be set and an interrupt generated, if conditioned. A1 will continue to acknowledge words transferred, but the data will not be stored.

(b) The A1 address counter reaches address 0000. If the 1604-B attempts to continue loading the AM, and an error condition exists, it is treated as in (a) above.

(c) Another EXF Select instruction is received by A1. This would occur, for example, if the (n) counter were presently too high for the number of words to be transferred, or the 1604-B program chose to abort the loading for some other operation. If this happens, an error condition exists and the underflow flip-flop will be set and an interrupt generated, if conditioned. The next EXF Select instruction referred to above cancels the incompleted LSL.

The γ field is used to indicate whether the tag bit is to be set to "1" or "0".

The first data word in all block load instructions (LSL, LPS) is assumed to be a mask word which is automatically loaded into the (m) register. All data is loaded through this mask.

LPS – Load Per S, α β γ

This block load instruction conditions A1 to load one or more 48-bit 1604-B words into certain locations of the AM array under control of the 16C4-B OUT instruction. The locations to be loaded are specified by the contents of the S register. The 1604-B instruction sequence is the same as that for the LSL load. Words will be loaded into successive locations, whose corresponding S register bits are the same as β , in order of decreasing addresses starting at the highest location 1023 or 2047 as selected by α . The tag bit associated with each word stored will be set to the same state as γ . The α field specified whether the S register is to be considered to correspond to the upper or lower half of the AM. Loading continues until one of the three same conditions as described under LSL occurs. For α specifying higher addresses condition, (b) will terminate at (a) = 1024.

LOS – Load One Word Per S

This instruction is intended to allow a single 48-bit word to be loaded into the AM array without resetting the response store scanner. This, along with the RNS instruction, will permit loops in the 1604-B which will process one response at a time rather than by blocks. The instruction is otherwise the same as the LPS and is terminated by conditions (b) or (c), but not (a) since the (n) counter is not used.

LDA – Load a, n Registers

This instruction conditions the AM to load one word into the (a) and (n) registers. Since both the (a) and (n) registers are only 11 bits for a total of 22, the remaining 26 bits in the 48-bit 1604-B word are ignored. The (a) register is loaded from the least significant 11 bits of the upper half of the 1604-B word and the (n) register from the least significant 11 bits of the lower half. The instruction sequence in the 1604-B would be the same as that for the LSL instruction. Since only one word can be transferred, it terminates after that word.

LDX – Load x Register

This is the same as LDA except that the comparand register is loaded instead of the (a) and (n) registers. Note that since the comparand register is also used as a transfer register, its contents are destroyed by any other load instruction.

LDM — Load M Register

This is the same as LDA except that the mask register is loaded instead of the (a) and (n) registers which is 48 bits rather than a total of 22 bits.

LDP — Load Prefix

This is similar to LDA except that only four bits are loaded into the Prefix register.

SOE — Search on Equality, α β γ

Execution of this instruction causes a comparison of the comparand and certain memory words in the bit positions specified by the mask register. If those words are identical in every bit position compared, the corresponding bit of the S register will be set to a "1". Since only one half of the memory can be searched at a time, the field is used to specify the half. The β and γ fields are used to specify what words are to be searched. If γ is a "0" the tag bit is ignored and all words are searched. If γ is a "1", the words are searched whose tag bits equal β.

SAE — Search on Equality, AND, α β γ

This is the same as SOE except that the results of the search are ANDed with the previous contents of the S register. This is equivalent to saying that only those locations are searched whose corresponding S register bit equaled "1" previous to SAE execution.

SOG — Search on Greater, α β γ

This is the same as SOE except that the search condition is "greater than" instead of "equals".

SGE — Search on Greater or Equal, α β γ

This is the same as SOE except that the search condition is "greater than or equals" instead of simply "equals".

RCS — Read Count of Responders, β

This instruction conditions the AM to read one word from the AM into the 1604-B under control of a 1604-B INT instruction. The sequence is similar to that of the LPS except that RCS initiates the scanner which may take as long as 16 microseconds to complete its counting (see Paragraph 2.5.6). If no word is actually transferred because the INT instruction is never executed, the instruction is terminated by the next EXF directed to AM. The underflow flip-flop is not set. The word read consists of an 11-bit code in the least significant part of a 1604-B word which is the count of the number of β 's in the S register; the prefix is suppressed.

RSS — Read Short Count

This is the same as RCS except that the count is terminated when +/S > 1 is realized.

RAS — Read Addresses of Responders, β

This instruction steps the scanner to the next response (downward) which conditions the AM to read one or more words from the AM into 1604-B under control of a 1604-B INT instruction. The sequence is similar to that for the LSL instruction including the conditions for termination. Note that this would mean that the (n) register would have to be loaded from the 1604-B to avoid setting the underflow or overflow flip-flop and generating an interrupt. A 15-bit address word is read with the four most significant bits coming from the prefix register, and the 11 least significant bits from the scanner.

RNS — Read Next Address, β

This instruction conditions the AM to read one word from the AM into the 1604-B under control of a 1604-B INT instruction. It is similar to the LOS instruction in that the response store scanner is not reset after each execution. Otherwise, it is similar to the RAS instruction.

IOE — Interrupt on Any Error

This instruction conditions the AM so that any error condition which sets an error flip-flop will also cause an interrupt to the 1604-B.

IIE — Inhibit Interrupts

This cancels the effect of IOE.

CHC — Channel Clear

This instruction is included for consistency with other 1604-B peripheral devices. It clears all interrupts, resets all sense flip-flops, terminates any outstanding instruction, and sets the response store scanner to address 2048.

CAM – Clear AM

This is the same as the CHC instruction, but the instruction can be addressed only to the AM in contrast with the CHC which is broadcast to all peripherai devices.

CIO – Clear Interrupt Only

This instruction resets the interrupt flip-flop and sense flip-flops.

SENSE INSTRUCTIONS

These instructions are shown in Table 2-2 and are considered self explanatory except that the α field is to be specified with each AM sense instruction to specify whether a "1" or "0" state is to cause an affirmative reply.

2.5.4 The A2 Configuration

2.5.4.1 Introduction. This paragraph describes the programming and functional details of the A2 configuration. The descriptions in the following paragraphs depend heavily on the foundation established in the preceding paragraphs of the A1 configuration. In addition, an understanding of the 1604-B programming is assumed; particularly the I/O system and the EXF, INT, and OUT instructions.

The A2 configuration is basically an elaboration of the A1; the response logic used is similar. While the associative and search functions of A2 are not much more powerful than those in A1, the design of A2 attempts to make the use and programming of the associative memory as flexible and easy as possible. Such a design will enable identification of the real value of the search functions in sea surveillance without the masking effects due to cumbersome set-up and control procedures. The A2 configuration is much more flexible and capable than A1 in the areas of control and the processing of successive responses. As with A1, A2 is attached to the 1604-B via channel 7. The following paragraphs give a general description of A2 and its modes of operation.

2.5.4.2 General Description of A2

(1) Registers and Data Flow. A register block diagram of A2 is shown in Figure 2-20. The figure also shows the possible data paths among the registers; number of bit positions in each register are shown in parentheses on each register. The registers are:

(a) The memory address (or address) register (MAR), which specifies the memory word to be read or written.

(b) The instruction (I) register, which may be loaded from the 1604-B and then used as a source of instructions for A2 operations.

(c) The memory buffer register (MBR), which serves as temporary storage for data read or written from A2.

(d) The prefix register (PR), which may be concatenated as the four high-order bits of an address from A2.

(e) The tag register (TR), which can be used in search or load instructions to specify tag operations.

(f) The tag mask register (TM), which masks the tag register in search and load operations.

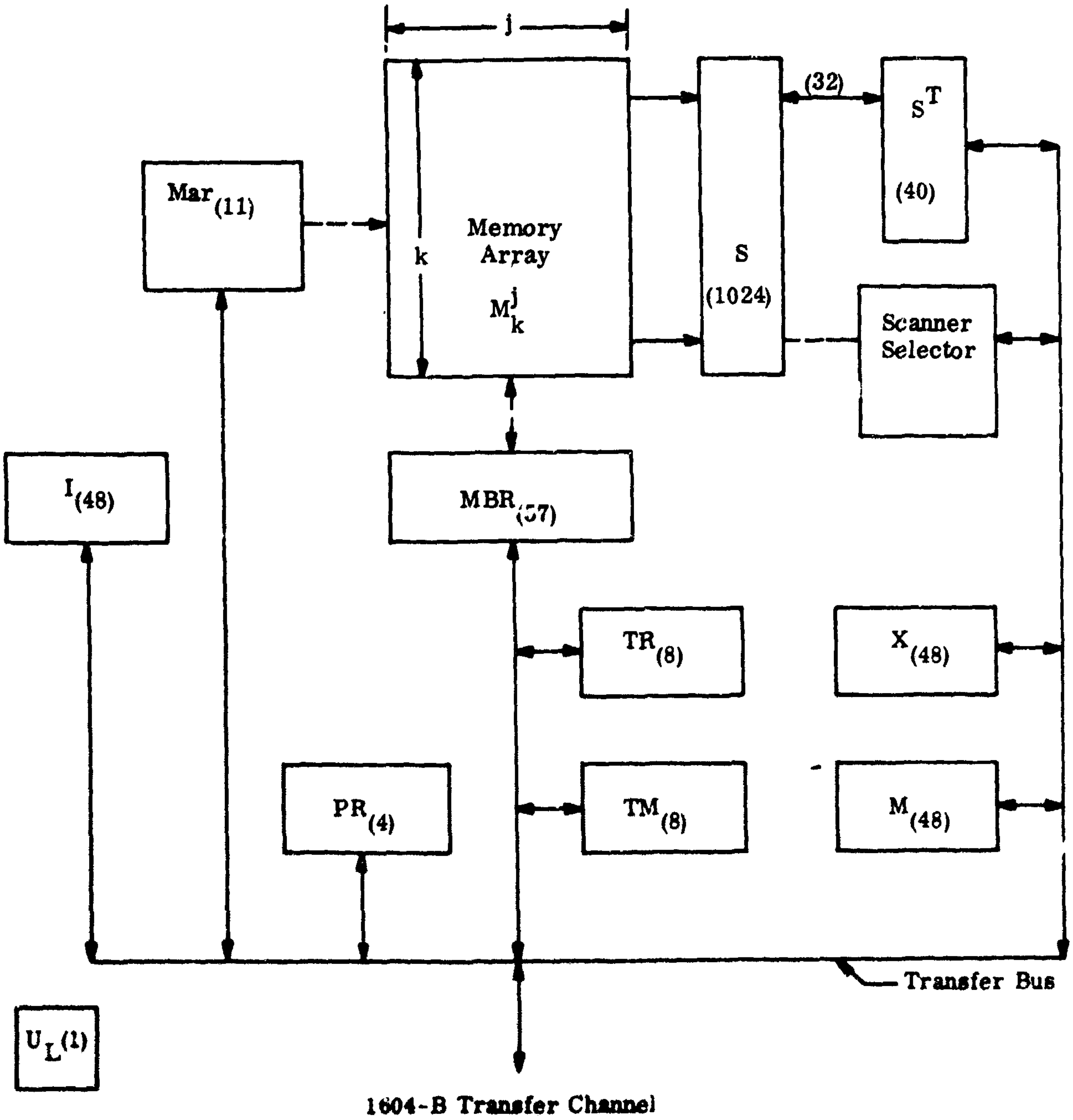
(g) The X register or comparand register, used to specify search arguments.

(h) The mask register (M), which masks the X register during searches, and also masks all data loaded into or read from A2.

(i) The S or sense response register, which contains the result of all search or logic operations at the end of such operation.

(j) The S^T (S Temp) register, which is used in processing logic operations on S.

NOTE: MBR and S^T are not directly available to the 1604-B.



$0 \leq k \leq 2047$

$0 \leq j \leq 56$

Figure 2-20. Register Block Diagram - Model A2

(2) Memory Array. The memory array is arranged into 2,048 words of 57 bits each and uses the Goodyear technology. Each word has 48 data bits, 8 tag bits, and 1 parity bit. Search capability exists for 1,024 words at a time, either the upper memory (locations 0000_{10}) or the lower memory (locations 1024_{10} through 2047_{10}). All data and tag bits may be read or written under 1604-B control and searched. Tag bits are treated differently from data bits as explained in the following paragraphs.

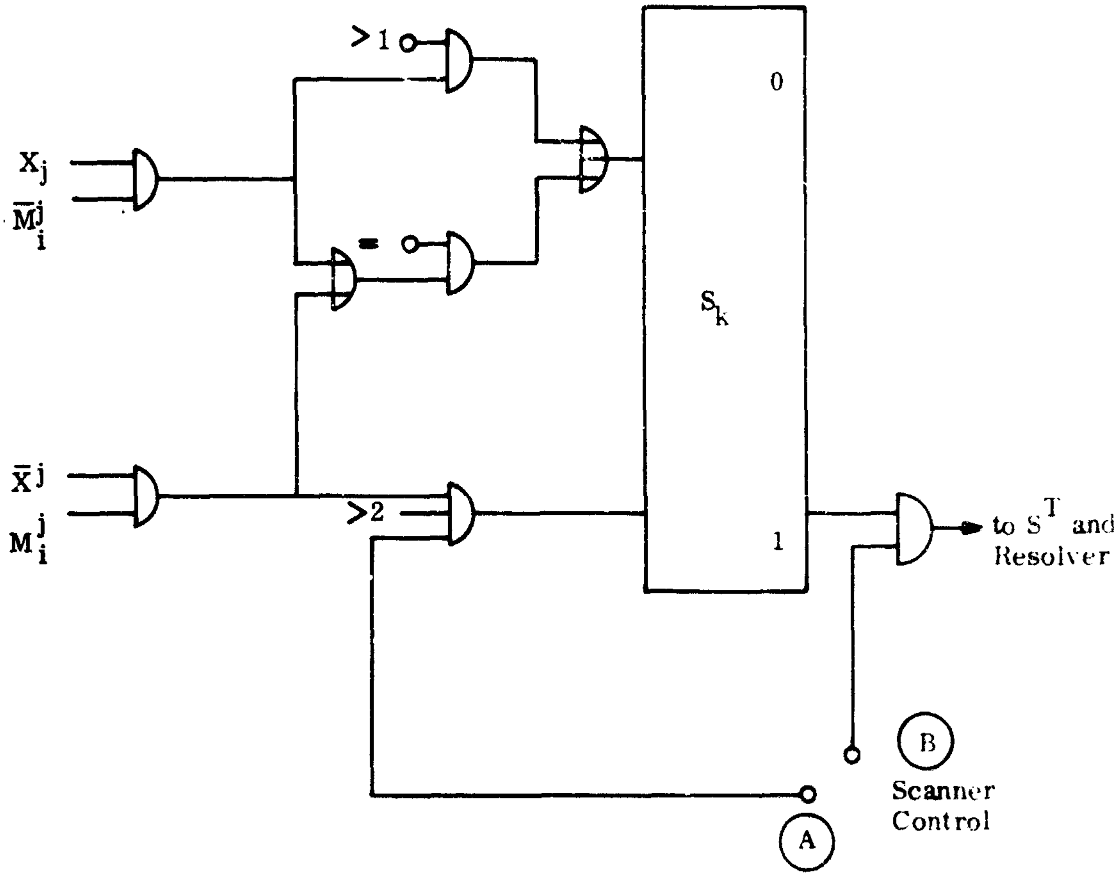
(3) S Register. The S register is a 1,024-bit register which normally receives the result vector of a search. When referring to a search result,the vector will be denoted simply by S. Response logic is associated with S. The logic is based upon the A1 machine, and the discussion of Paragraphs 2.5.3 also applies here. In addition to the logic of A1, the A2 machine adds the capability to form any Boolean function of S and a set of 1,024 tag bits in memory; or of S and 1,024 bits supplied by the 1604-B; or of S and the result of a search. In addition, S may be shifted up or down. A cross section of the S logic is shown in Figure 2-21.

To form a Boolean logic function between S and one of the other arguments just mentioned, S is treated as thirty-two 32-bit registers. The function is formed, in 32 steps, by reading a segment of S into S^T and performing the necessary logical and shifting operations in S^T; S^T is then gated back to S. The second argument for these operations is obtained by a readout of tag bits from the memory or by search (both results appear in S);or by the transfer of 32 sets of 32 bits from the 1604-B. In all cases, the result is formed in S^T and transferred into S.

Referring to Figure 2-21, the gate inputs marked ">1", ">2", and "=" are enabled in groups of 32 to permit selective loading of S. The gate inputs marked A and B permit the transfer of information between S and S^T.

(4) Tag Facilities. The memory array provides eight bits per word in addition to the usual 48 data bits which are tag bits. These bits are used for loading under control of the 1604-B and may be searched. In addition, S may be stored in a set (column) of tag bits.

The tag and tag mask registers may be used in load and in search operations. When loading data into the associative memory, the 1604-B program may specify that either the tag bits remain unchanged, the TR be copied through TM into the tag bit positions, or a specific tag bit be set or reset.



Parts Count

Number of Gates - 8
Number of Lines - 17
Number of Instructions - 8
Gates/Instruction = 1
Lines/Instruction = 2.12

Figure 2-21. S Register Logic, One Stage

In like manner, a search may specify that the bit configuration in TR, masked by TM, be matched, a specific tag bit be set or reset, or the tag bits be ignored. Tag bit control is fully explained in Paragraph 2.5.4.4.

(5) Logic Abilities

(a) Logic with the 1604-B. The contents of the S register may be read into the 1604-B. In reading the contents of S, the 1604-B accepts 32 words, each word 32 bits. Using S^T, any logical function of two variables may be generated in S, on a bit-by-bit basis, between the contents of S and a 32-word block of data supplied by the 1604-B. The function is actually generated in S^T in 32 steps, with the partial results placed in S at each step. The functions are given in Table 2-3. In the table, R denotes the bit supplied by the 1604-B and the S bit in S at the start of the operation. Note that this logic capability allows the setting of S from the 1604-B (F3). The time required is a function of the logic operation specified. For trivial operations (FØ, F5, F1Ø, F15), essentially no A2 time is needed; if an output is generated by the 1604-B, the 32 words require [4.0 + (4.8 x 32)] = 158 microseconds for transfer. All non-trivial logic operations are completed during the output of data so 158 microseconds is the maximum time.

(b) Logic on Search Operations. By using the SLS (Search Logic Specification) instruction,the programmer may form, in S, any logical function of two searches. Note that a given search may only involve the tag bits, if desired, or the short instruction forms (eight bit instructions) may be used for tag bit logic. Search operations which require logical manipulation of S (other than a simple complement such as F1Ø) are performed as follows. (Remember that S may be considered as thirty-two, 32-bit registers.) Each search is performed 32 times, each time on a different segment of S by using the inhibit and select ability added to the A2 S logic. Using S^T as a temporary store and logic register, the full S vector is developed in 32 steps; thus, search time for a search modified by SLS is 32 x 5 microseconds = 160 microseconds.

TABLE 2-3. FUNCTIONS FOR LOGICAL OPERATIONS

n	Algebraic Function	Name or Comment
0	0	ZERO → S
1	RS	AND
2	RS'	
3	R	Simple Search, No connective
4	$R'S$	
5	S	
6	$R \oplus S = R'S+RS'$	Exclusive OR
7	$R+S$	OR
8	$R \downarrow S = R'S' = (R+S)'$	Peirce Stroke, NOR
9	$(R \oplus S)' = R'S' + RS$	
10	S'	
11	$R+S'$	
12	R'	
13	$R'+S = R \rightarrow S$	Implication
14	$R \mid S = R + S' = (RS)'$	Sheffer Stroke, NAND
15	1	ONE → S

NOTE: 1. R = The value derived from an interrogation or search of memory, or the value supplied by the 1604-B.

2. The binary value of the designated algebraic function appears in the S register at the end of all logical operations.

2.5.4.3 Principles of Control

(1) Instructions. A2 instructions are 12 or 8 bits in length, and may be produced as EXF codes on channel 7 or as bit fields in the I register. Many instructions specify what action the A2 machine is to take with the data supplied by a subsequent OUT or INT instruction executed in the 1604-B. In these cases the conditions established will obtain until the action is taken or the conditions changed by additional instructions to A2. For example, if a LSL (load specific location) instruction is given to A2, the next OUT instruction in the 1604-B will cause the loading of the block of data supplied, unless some other A2 instruction changes the set-up. In this example, a LLR (load long register) would change the effect of the next OUT but a RSL (read specific location) or an SOE (search on equal) would not.

(2) Instruction Formats. This description assumes the existence of a suitable assembly language for the 1604-B A2 configuration. Thus, instructions are given as mnemonic operations with modification fields, and the assembler must produce correct bit encoding of the instructions.

(3) Instruction Fields. A number of specification fields are used in the A2 instructions. The field symbol, the list of valid entries, and the interpretation are given in Table 2-4.

2.5.4.4 Description of Instructions. This paragraph describes all A2 instructions. The fields employed are described in Table 2-4. The heading line of each instruction gives its mnemonic, the fields that must be specified, and the instruction name.

Instructions may be presented to A2 as 12-bit EXF codes on channel 7 or as a 12 or 8-bit field in the I register. Control is transferred to I by the XIS or XIL instructions. When transferred, instructions in I are executed in order, left to right. While executing instructions in I, A2 ignores EXF code instructions.

(1) Load Instructions (12-bit format)

LLR   X,M,A,I   Load Long Registers

Loads* the designated registers in the order X,M,A,I. If the OUT sends too many or too few words, an overflow or underflow condition is established. The OUT should transmit one word for each designated register with the data right justified.

* Note that LLR does not load the registers; this is accomplished by an OUT instruction (see Paragraph 2.5.4.3, item (1)).

TABLE 2-4. INSTRUCTION FIELDS

Field Symbol	Valid Entries	Interpretation
X	X	The X register is designated by the instruction.
	-	The X register is not designated by the instruction.
M	M	The M (mask) register is designated by the instruction.
	-	The M (mask) register is not designated by the instruction.
A	A	The A (address) register is designated by the instruction.
	-	The A (address) register is not designated by the instruction.
I	I	The I (instruction) register is designated by the instruction.
	-	The I (instruction) register is not designated by the instruction.
PR	PR	The PR (prefix) register is designated by the instruction.
	-	The PR (prefix) register is not designated by the instruction.
TR	TR	The TR (tag) register is designated by the instruction.
	-	The TR (tag) register is not designated by the instruction.
TM	TM	The TM (tag mask) register is designated by the instruction.
	-	The TM (tag mask) register is not designated by the instruction.
TH (Tag Handling)	IT	Ignore tag bits in searching/Retain present tag bits in loading.
	TR	Use tag register when searching/Load tag register when loading.
	SZ	When searching, search the tag bit designated by Y for ZERO/ When writing, set the tag bit designated by Y to ZERO.
	SO	When searching, search the tag bit designated by Y for ONE/ When writing, set the tag bit designated by Y to ONE.
	-	Same as IT.
Y	Ø	Designates one of tag bits Ø-7 (for use with the TH field).
	1	
	2	
	3	
	4	
	5	
	6	
	7	
	-	Same as Ø

NOTE: The symbol "-" denotes absence of an entry.

TABLE 2-4. INSTRUCTION FIELDS (Cont.)

Field Symbol	Valid Entries	Interpretation
UL	U	Specifies that the operation is to apply to the upper half of memory (locations 0000_{10} through 1023_{10}).
	L	Specifi-s that the operation is to apply to the lower half of memory (locations 1024_{10} through 2047_{10}).
TF	T	Specifies the TRUE state of S_1 for use in the operation (ONES).
	F	Specifies the FALSE state of S_1 for use in the operation (ZEROS).
	-	Same as T.
IPT	I IPT	Causes an interrupt at the end of the operation.
	NI NIPT	Does not cause an interrupt at the end of the operation.
	-	Same as NI.
STEP	S STEP	Causes the scanner to step to the next response after the operation.
	-	No effect.
P	P	Causes the contents of the prefix register to be appended to the address read by the operation.
	-	No effect.
BA	B	Causes the shift of S specified by the SHIFT field to occur before the operation.
	A	Causes the shift of S specified by the SHIFT field to occur after the operation.
Logic OP	FØ F1 ⋮ F15	Specifies a logical algebraic function as specified by Table 2-3.
	-	Same as F3 in Table 2-3.
SHIFT	-3 -2 -1 1 2 3 4	Specifies the number of places and direction of shifting for S (a negative shift is up, high to low).
	-	Same as Ø

(1) Load Instructions (Cont.)

LSR PR, TR, TM Load Short Registers

Identical to LLR, except for registers designated.

LTR [8-Bit Field] Load Tag Register

The 8-bit field in the instruction is copied into the tag register.

LTM [8-Bit Field] Load Tag Mask Register

The 8-bit field in the instruction is copied into the tag mask register.

LPR [4-Bit Field] Load Prefix Register

The 4-bit field in the instruction is copied into the prefix register.

LSI TH, Y Load Specific Locations

The data supplied by an OUT will be loaded into consecutively decreasing memory locations starting with the present contents of the address register. The TH and Y field specify the loading of tag bits. Location 2047_{10} follows location 0000_{10} for this instruction.

LPS TH, Y, UL, TF, IPT Load Words per S

The data supplied by an OUT will be loaded into decreasing memory locations in the upper or lower (selected by UL) memory half which satisfy the S condition selected by TF. The first word is added to the highest address which satisfies TF in UL. The TH and Y fields specify the loading of tag bits. If interrupt is selected by the IPT field, it will be generated when the OUT attempts to load more words than S can accept and when the OUT does not exhaust S.

LOS TH, Y, UL, TF, STEP Load One Word per S

The data supplied by an OUT will be loaded into the present (or first) memory location in U or L selected by TF and S just as in the LPS instruction. If more than one word is transferred by an OUT, an overflow error is generated. If STEP is selected, the scanner steps to the next selected (TF) S position.

NOTE: In LSL, LPS, and LOS instructions all data bits are loaded through a mask contained in the mask register while the tag bits, if TR is selected, are loaded through the tag mask register.

(2) Read Instructions (12-bit format)

RLR X,M,A,I Read Long Registers

Reads the designated registers in the order X,M,A,I. If the INT calls for too many or too few words, an overflow or underflow condition is established. The INT should call for one word for each designated register; the data will be right justified.

RSR PR, TR, TM Read Short Registers

Identical to RLR except for registers designated.

RSL Read Specific Locations

The data called for by an INT instruction will be taken from consecutively decreasing memory locations starting with the contents of A. Location 2047$_{10}$ follows location 0000$_{10}$ for this instruction. Data is read through the mask register.

RAS P,UL,TF,IPT Read Addresses per S

An INT reads 15-bit right justified address corresponding to positions in S selected by TF. Addresses are read in decreasing order. The 10 low-order bits are derived from S, the eleventh bit corresponds to the UL selection, the four high-order bits are supplied by the prefix register, if selected, or else ZEROs are filled in. If IPT is selected, an interrupt is generated when the last valid address in S has been transferred.

RNA P,UL,TF,STEP,IPT Read Next Address

The same as RAS except: (1) the INT reads the current (or first) address selected by TF and S. (2) If STEP is selected, the scanner steps to the next S position after the address is read. (3) If IPT is selected and there are no more positions in S to satisfy TF, an interrupt will be generated after A2 supplies a dummy address of 0000$_{10}$. If the INT calls for more than one word or an unavailable word, all words after the first will be all ZEROs and an overflow condition will be established.

RPS UL,TF,IPT Read Words per S

An INT reads words from decreasing memory locations which satisfy the conditions of UL,TF, and S. If IPT is selected, an interrupt is generated after the last word is transferred. If the INT calls for more words than are available, an overflow condition is established and words of all ZEROs are transferred to satisfy the INT. Data is read through the mask register.

ROS UL,TF,STEP,IPT Read One Word per S

An INT reads the current (or first) word which satisfies UL,TF, and S through the mask register. If STEP is selected, the scanner steps to the next (decreasing) memory position which satisfies the conditions after the operation. If IPT is selected, an interrupt is generated after transfer of an all ZERO word after the last valid word has been transferred. If the INT calls for more than one word or for an unavailable word, all words after the first are ZERO and an overflow condition is established.

RCS TF,IPT Read Count of S

A count is taken of the positions of S which satisfy TF. If IPT is selected, an interrupt is generated after the count is developed. An INT will read the count, right justified.

RSS TF,IFT Read Short Count of S

A count is started of the positions of S which satisfy TF. The count is terminated if it reaches two, otherwise the same as RCS.

RSR Read S Register

An INT of 32 words reads the S register in thirty-two 32-bit segments; each segment is right justified. The first word read contains bits 0-31 of S with bit 31 in the right-most position. The last word read has bit 1023 of S in the right-most position.

RTS Read Tags Specific

An INT reads the eight tag bits of successively decreasing memory locations starting with the location addressed by A through the tag mask register. Location 2047_{10} follows location 0000_{10}. The bits are left justified.

(3) Search Instructions (12-bit format)

SOE TH,Y,UL Search on Equal

The S register is set to ONE in all positions for which the unmasked parts of the word in UL exactly match the X register as masked by M and the requirements of TH and Y are satisfied; all other positions are set to ZERO.

SAE TH,Y,UL Search on AND Equal

The S register will have a ONE in all positions which satisfy the SOE search and which were already set to ONE as a previous result.

SOG TH,Y,UL Search on Greater

The S register is set to ONE for each position in UL in which the unmasked bits of the word are greater than the corresponding bits of X. The unmasked bits are treated as a concatenated diminished radix complement (ONE's complement) field. Only those words whose tag bits satisfy TH and Y are set; all other positions of S are set to ZERO.

SGE TH,Y,UL Search on Greater or Equal

Similar to SOG except the condition is greater or equal.

SLS Logic OP,BA,SHIFT,IPT Search Logic Specifications

When this instruction immediately precedes one of the search instructions, it modifies the execution of the instruction and provides logical and shifting capabilities between searches. The effect of the SLS is:

(a) If the logic OP selected is not trivial (such as FØ, F5, etc.) the search is performed in 32 steps (in 160 μs), employing the S^T register to hold temporary results. The search forms in S the algebraic function of the specified search and the previous contents of S, see Table 2-3. (R denotes the results of the current search in the table.)

(b) If a SHIFT is specified, it occurs before or after the search operation is selected by the BA field. A positive shift is down, bits of S shifted off the end of S are lost, bits are filled with ZERO at the beginning.

(c) If IPT is selected,an interrupt will be generated at the conclusion of the operation.

(4) Execute and Logic Instructions

XLI [8-bit Field] Execute Logic Immediate

This instruction causes the execution of the eight-bit instruction contained in the field.

XLC Logic OP, IPT Execute Logic with Computer

The Logic OP specified is formed in S between an S (like) vector supplied by the 1604-B (32 words of 32 bits) and the contents of S. If IPT is selected,an interrupt is generated at the end of the operation.

XIS — Execute Instructions Short

This instruction transfers control of A2 to the I register which must contain a string of eight-bit instructions. The instructions are performed in order from left to right.

XIL — Execute Instructions Long

Same as XIS, except 12-bit instructions are in I.

(5) Eight-Bit Instructions

These instructions can be executed from the I register or as a result of the XLI instruction.

LO — Logic OP, Y — Logical Operation

The specified Logical OP is performed between the tags specified by Y in U or L, as last specified by some instruction. The result appears in S.

SO — SHIFT, Y — Shift Operation

S is shifted as specified by SHIFT and stored in the tag bits specified by Y and the last specified UL.

CL — Logic OP — Computer Logical Operations

The same as XLC, except for INT.

CS — SHIFT — Computer Store

The same as RSR, except the specified shift of S is taken before the store operation.

SH — SHIFT — Shift S

The S register is shifted as specified.

IPT — Interrupt the 1604-B

Interrupts the 1604-B.

NO — No Operation

A pass instruction.

(6) Miscellaneous Control Instructions

IOV	Interrupt on overflow
IUV	Interrupt on underflow
IER	Interrupt on error
CAM	Clear AM
CIN	Clear all interrupt conditions
CLE	Clear all error and over or under flows
CLR	Clear all AM registers
CIS	Clear interrupt selections

(7) Sense Operations

TRA	Sense alive
TRY	Sense ready
TRE	Sense error
TRO	Sense overflow
TRV	Sense underflow
TES	Sense +/S = 1
TLS	Sense +/S < 1
TGS	Sense +/S > 1
TSE	Sense scanner at end.
TIP	Sense interrupt

2.5.5 The A3 Configuration

2.5.5.1 Introduction. The following paragraphs describe the A3 configuration utilizing the previous descriptions of the GAP, A1, and A2 configurations.

A3 is a fully integrated configuration in which the AM becomes an integral part of the 1604-B organization — replacing, and acting as, the highest numbered 2,048 words of 1604-B memory. The 1604-B may be programmed normally if desired, since the integration of the AM retains all normal 1604-B operations.

The elaborate and complex GAP-DMA configuration suffers from its architecturally ambiguous position as a semi-multiprocessor peripheral device. That is, the overhead in the 1604-B necessary to set up, control, coordinate, and synchronize the GAP (operating on the DMA channel) is very high. A3 eliminates most of this overhead by allowing direct manipulation of the associative features of the AM by new 1604-B instructions, and by allowing direct I/O between the AM and the peripheral devices. A3 has also generalized the equipment of the GAP to allow more flexible and powerful programming. The AM itself is based upon the GAP and is very similar to it.

2.5.5.2 Configuration and Organization of A3. The 1604-B memory is composed of two banks of 16,384 words each. One bank contains all even addresses and the other all odd addresses. In A3 a third bank is added, the AM itself, which contains all addresse greater than 30,719. To the user of 1604-B instructions and I/O channels there is no change in 1604-B operation. Of course, the last 2,048 memory locations have considerabl extra capability since they are an AM. The logic associated with the associative functionir of the AM and with the A3 configuration is shown in Figure 2-22. AM organization is a generalization of the associative features of the GAP as explained in the following paragrap

Referring to Figure 2-22, the following registers and logic networks are shown:

S^1, S^2 : 1604-B memory address registers (14 bits) for the odd and even banks of 1604-B core.

Z^1, Z^2 : 1604-B memory buffer registers (48 bits) for the odd and even banks of 1604-B core.

X, A, Q : 1604-B general and arithmetic registers (48 bits) of the 1604-B.

AMAR : Memory address register (11 bits) for the AM.

AMBR : Memory buffer register (50 bits) for the AM.

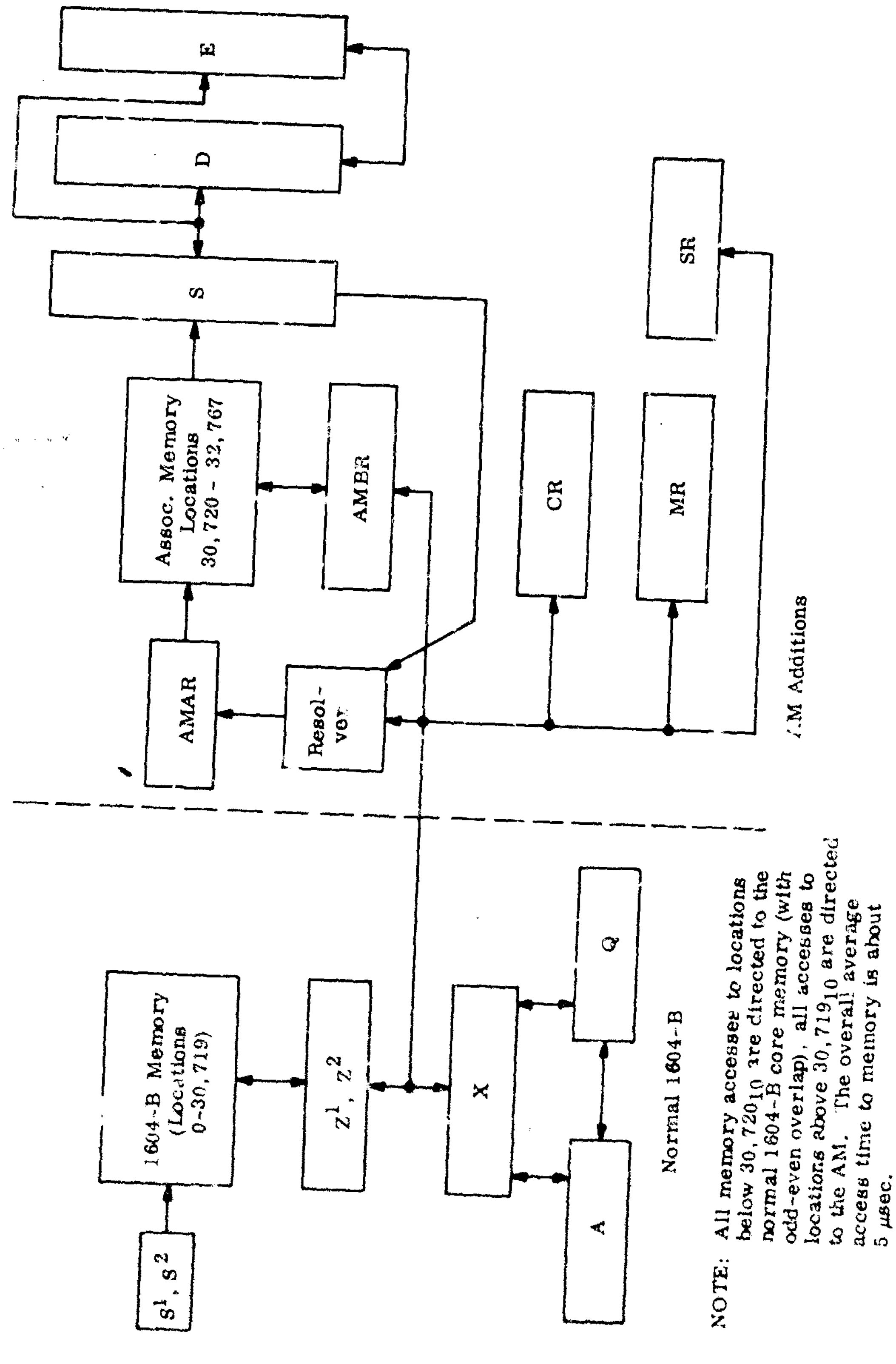
CR : Comparand register (49 bits) holds a search argument (including busy bit).

MR : Mask register (49 bits) used to specify the bits of memory to search or to control masking of words passing through AMBR when required.

SR : A storage register and counter combination (7 bits each) which control the shifting of words passing through AMBR or of the contents of CR when required.

S : A 1,024-bit register which receives the result of all searches, sometimes called a vector.

D, E : 1,024-bit registers which may hold a response vector.

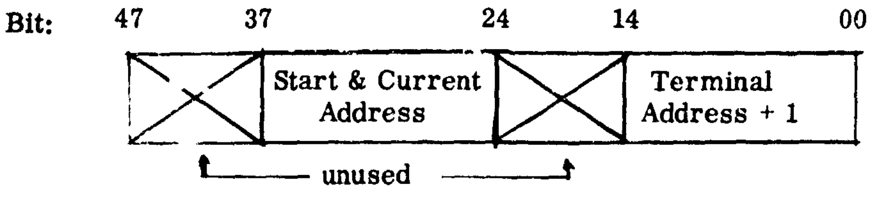


NOTE: All memory accesses to locations below $30,720_{10}$ are directed to the normal 1604-B core memory (with odd-even overlap), all accesses to locations above $30,719_{10}$ are directed to the AM. The overall average access time to memory is about 5 μsec.

Figure 2-22. A3 1604-B Associative Memory Integrated Configurations

Resolver : A counter and logic structure which can generate a count of the number of "ONES" or "ZEROS" in S, D, or E or which can develop a 15-bit binary number whose low order 10 bits correspond to a position of S, D, or E as selected by instruction.

2.5.5.3 <u>Input-Output</u>. In the 1604-B buffered I/O is controlled by control words held in memory. The format of a control word is:



In the context of the 1604-B, the terminal address must be greater than the starting address and less than or equal to 32,768 (actually zero). That is, the I/O operation must not wrap around from the highest memory location to the lowest. This is because the 1604-B uses memory locations 0 through 7 for special control functions. <u>All normally legal 1604-B I/O operations operate unchanged in A3.</u>

If the high order four bits of the starting address designate the AM (i.e., = 1111_2) and the terminal address is valid (i.e., in the range 111100000000001_2 through 000000000000000_2 MOD2^{15}), operations are normal. However, if the terminal address is invalid for normal 1604-B operation, it will designate special associative I/O functions as defined in the following paragraphs.

The facilities provided in the A3 I/O system allow:

(1) Normal 1604-B operation.

(2) Block loading of AM with busy bit control.

(3) Unloading or loading of a set of words which are defined by one of the response vectors, i.e., associative transfers.

The I/O control logic in the A3 configuration will examine bits 14 and 11 of the control word when bits 37 and 34 are all "1"s to determine the kind of operation to perform. There are three classes of operation: normal I/O in which the busy bit in AM is left unchanged, block associative I/O which provides control over the busy bit during a block load operation, and vector or full associative I/O. The classes are selected by bits 14 and 11 as follows:

Bits		
14	11	Operation
0	0	Normal
0	1	Vector
1	0	Block
1	1	Normal

If vector or block operation has been selected, bits 13 and 12 control the busy bit on writing to memory as follows:

Bits		
13	12	Operation
0	0	Do not change BB
0	1	Invert BB
1	0	Write BB as ONE
1	1	Write BB as ZERO

In block associative operations when bits 10 through 0 of the control word define the terminal address in the AM section of core, then the operation is similar to normal 1604-B I/O. In vector operations bits 10 through 0 are used internally for special control. The words to be transferred are defined by the combination of a response vector, the true or false positions of the vector, and the upper or lower addresses of AM as selected by bits 33 through 30. The bit configurations are:

	Bits			
33	32	31	Vector	Memory Block
0	0	0	E	-
0	0	1	E	U
0	1	1	D	L
0	1	0	D	U
1	1	0	DE	LU
1	1	1	DE	UL
1	0	1	ED	UL
1	0	0	ED	LU

The notation DE or UL denotes the concatenation of the vectors or address blocks indicated in the order written. Thus, the code 101 causes E to map the upper memory and D to map the lower memory. Bit 30 selects the TRUE state of the vector if 1, the FALSE state if 0. Upper memory addresses are 30,720 through 31,743; lower addresses are 31,744 through 32,767

2.5.5.4 A3 Instructions

(1) Normal 1604-B Instructions. All 1604-B instructions operate as defined in the 1604-B programming manual, except the two illegal instructions 77_8 and 00_8. Instead of unconditional halts these instructions are used to control the associative features of A3.

(2) 1604-B-Like Instructions. The 77_8 operation code in A3 provides for the use of all operand-type normal 1604-B instructions using associatively-defined operands. Instructions may be executed from the upper or lower half-word. The format of these instructions (shown in the upper half-word) is:

Bit:

47 42	41 39	38 33	32 29	28	27	26	25	24
77	b	Opcode	T	STEP	INT	MASK	SHIFT	BB

The fields of the instruction are:

Bits	Use
47-42	77_8 1604-B Opcode denoting this instruction class.
41-39	Index-Indirect field, operates on an associatively-defined (by T-field) 15 bit number, y, as in the normal 1604-B.
38-33	Sub-opcode — selects the instruction. See Table 2-5.
32-29	T-field causes the development of a 15-bit number from a combination of the S, D, and E vectors, the TRUE or FALSE state, and the upper or lower halves of the AM. This number is used just as the y (base execution address) is used in normal 1604-B operation.

TABLE 2-5. OPCODES FOR 77_8 INSTRUCTIONS

04	ENQ	Enter Q
10	ENA	Enter A
11	INA	Increase A
12	LDA	Load A
13	LAC	Load A Complement
14	ADD	Add
15	SUB	Subtract
16	LDQ	Load Q
17	LQC	Load Q Complement
20	STA	Store A, copy bit 24 into BB
01	STA	Store A, invert BB if bit 24 = 1, retain BB if bit 24 = 0
21	STQ	Store Q, copy bit 24 into BB
02	STQ	Store Q, invert BB if bit 24 = 1, retain BB if bit 24 = 0
22	AJP	A Jump
23	QJP	Q Jump
24	MUI	Multiply Integer
25	DVI	Divide Integer
26	MUF	Multiply Fractional
27	DVF	Divide Fractional
30	FAD	Floating Add
31	FSB	Floating Subtract
32	FMU	Floating Multiply
33	FDV	Floating Divide
36	SSK	Storage Skip
37	SSH	Storage Shift
40	SST	Selective Set
41	SCL	Selective Clear
42	SCM	Selective Complement
43	SSU	Selective Substitute
44	LDL	Load Logical
45	ADL	Add Logical
46	SBL	Subtract Logical
47	STL	Store Logical, copy bit 24 into BB
03	STL	Store Logical, invert BB if bit 24 = 1, retain BB if bit 24 = 0
50	ENI	Enter Index
51	INI	Increase Index
52	LIU	Load Index, U
53	LIL	Load Index, L
55	IJP	Index Jump
56	SIU	Store Index, U
57	SIL	Store Index, L
60	SAU	Substitute Address, U
61	SAL	Substitute Address, L
64	EQS	Equality Search
65	THS	Threshold Search
66	MEQ	Masked Equality
67	MTH	Masked Threshold
70	RAD	Replace Add
71	RSB	Replace Subtract
72	RAO	Replace Add One
73	RSO	Replace Subtract One
75	SLJ	Selective Jump
76	SLS	Selective Stop
77	NOP	No Operation, however the Step and T fields are effective

Bits	Use
28	Step — cause the resolver to step to the next or first response before developing the base execution address, y.
27	Interrupt — causes a program interrupt if there is no next or first response as defined by T when called for by Step.
26	Mask — causes the word passing through AMBR to be masked by MR.
25	Shift — causes the word passing through AMBR to be right circular shifted the number of places held in SR before any masking.
24	Busy Bit — control is applied only to the 77 class instruction codes 20_8, 21_8, and 47_8. See the discussion in Table 2-5.

The T field (and the b field) control the development of an operand address. Assume that as the result of some search sequence, a response vector is established and is stored in the S, D, or E registers. The vector in fact is a mapping of the U or L part of AM, and either the TRUE or FALSE state of the bits of the vector may give the correct mapping. When an associatively-defined operand is required, a 15-bit number, y (the base execution address), must be developed. The high-order four bits of y are 1111 since the base execution address is in the AM; the low-order 10 bits are the result of resolving the S, D, or E vector in the T or F state as selected by the T field. The missing bit (2^{10}) is selected by the T field to match the U or L designation. The combinations of S, D, E; T, F; and U, L selected by the T field are:

T Bits 32 - 39	Vector	T, F	U, L
0000	S	F	U
0001	D	F	U
0010	E	F	U
0100	S	F	L
0101	D	F	L
0110	E	F	L
1000	S	T	U
1001	D	T	U
1010	E	T	U
1100	S	T	L
1101	D	T	L
1110	E	T	L

Using this set of instructions, the programmer may directly access associatively-defined operands. He may index them or use them as indirect address by proper coding of the b, or index, field.

When operating with a set of associative operands, the selected vector is resolved from the S register. The transfer of this vector to the S register for resolution is handled automatically by the A3 logic without destroying any response vector (S, D, or E). The resolver may be thought of as a counter which steps down the vector until the proper condition (T or F) is found. Storage is provided in the resolver to allow the I/O system to resolve a vector when transfer is needed without losing its place in normal operations. Between I/O transfers, the position of the I/O vector resolution is stored in bits 10-0 of the I/O control word. In using the 1604-B instruction in the associative mode, the programmer must not change vectors unless he is willing to lose his place in the old vector. The A3 logic will step properly through one vector. If the programmer desires to change vectors, he should store, and later reset, the resolver, using the instructions described in Paragraph 2.5.5.4, item (4).

(3) Search Instructions. A3 provides the 10 search instructions of the GAP,* plus extensive detail control over the inter-search logic and vector manipulation.

In addition to the GAP search instruction, variations of MAX, MIN, NHC, and NLC are provided which do not require the use of two response registers. In the GAP, the D and E registers are associated with the upper and lower halves of memory; in A3 they are more general. This generality is provided through the detail control of a search available in A3.

All searches place a result in the S register; however, S may be exchanged with D or E both before and after the search to allow (effectively) the placing of results directly into D, E, or S.

The format of an A3 search instruction is shown below in the upper half word:

Bit:	47	42	38	37	36	35	34	33	32	31	30	29	28	27	24
	00_8	OP		UL	SD	SE	DS	ES	INIT	ADV	SH	MASK	BB	LOGIC	

*See Appendix A.

The fields of instruction are:

Bits	Use
47-42	00_8 — 1604-B Opcode designating this instruction set.
41-38	Sub-Opcode — see Table 2-6.
37	UL — Selects a search on upper or lower part of AM.
36	SD — Exchange S & D before the operation.
35	SE — Exchange S & E before the operation.
34	DS — Exchange D & S after the operation.
33	ES — Exchange E & S after the operation.
32	INIT - Initialize S before the search.
31	ADV - Advance S before the search.
30	SH — Shift the comparand right circular (on SR) before the search.
29	MASK — Use the MR to mask CR.
28	BB — Ignore the BB in this search.
27-24	LOGIC — Perform the logical operation specified by Table 2-7 after all other operations in the search.

(4) Control Instructions. A set of control instructions is provided in A3 to manipulate the additional facilities. These instructions have three formats as follows (shown in upper half word):

	47	42 38	37	36	35 30	29	28	27	26	25 24
F1	00_8	15_8	0	0	W	SH	MASK	ADVAN	INT	B B

	47	42 38	37	36	35 30	29 26	25 24
F2	00_8	15_8	1	0	W	OP	☒

	47	42 38	37	36	35 26	25 24
F3	00_8	15_8	1	1	OP	B B

TABLE 2-6. SEARCH INSTRUCTIONS

EMC	Exact Match
MMC	Mismatch
GEC	Greater than or Equal
GTC	Greater than
LEC	Less than or Equal
LTC	Less than
MXF	Maximum Fast – the register selected by bits 35, 36,or the S register is destroyed by the operation; the other register contains the result.
MXS	Maximum Slow – no register is destroyed (except the result, of course).
MNF	Minimum Fast – the register selected by bits 35, 36,or the S register is destroyed by the operation; the other register contains the result.
MNS	Minimum Slow – no register is destroyed (except the result, of course)
NHF	Next Higher Fast – see MXF.
NHS	Next Higher Slow – see MXS.
NLF	Next Lower Fast – see MXF.
NLS	Next Lower Slow -- see MXS.
NOP	No search, all other controls are active.
EXT	Extend – this opcode defines the control instructions.

TABLE 2-7. LOGIC FOR SEARCH OPERATIONS

Logic Field	Operation
0	S + D → S
1	S + E → S
2	S + D + E → S
3	0 → S
4	1 → S
5	S + D → D
6	S + E → E
7	SD → D
8	SE → E
9	S → D
10	S → E
11	0 → D
12	0 → E
13	1 → D
14	1 → E
15	No Op

The fields of the instruction are:

Field	Use
Bits 47-36	Designate the control instructions and the subclass
OP	Opcode for the subclass; note subclass 00 has only one instruction
W	A field which defines a combination of T, F: S, D, E: and U, L to designate a set of words in memory. See Table 2-8.
BB	Busy bit control when writing to memory or loading MR or CR, see Table 2-9.
SH MASK ADVAN INT	As in other instructions.

The F1 instruction is:

WCR — Write Constant into responders. All control fields are operative.

The F2 instructions are:

RCA — Read Count of responders to A

RCQ — Read Count of responders to Q

The F3 instructions with BB effective are:

LMA — Load MR from A

LMQ — Load MR from Q
LAM — Load A from MR
LQM — Load Q from MR
LCA — Load CR from A
LCQ — Load CR from Q

LAC — Load A from CR

LQC — Load Q from CR

The F3 instructions without BB effective are:

LSA — Load SR from A

LSQ — Load SR from Q

LAS — Load A from SR

LQS	—	Load Q from SR
LRA	—	Load Resolver from A
LRQ	—	Load Resolver from Q
LAR	—	Load A from Resolver
LQR	—	Load Q from Resolver

(5) Miscellaneous 1604-B Instructions. Additional 1604-B instructions have been defined for A3. These instructions are "internal" EXF sense and select codes to allow testing and setting of conditions in the AM hardware. They are:

SIP	Select Interrupt on Parity Error
CPI	Clear Parity Interrupt
CPS	Clear Parity Interrupt Selection

TCZ	Skip if count equal zero	and complementary conditions
TCO	Skip if count equal one	
TCM	Skip if count greater than one	
TMB	Skip if memory busy	
TMI	Skip if memory interrupt	

TABLE 2-8. W-FIELD CONTROL

1. W is a six-bit field: W_5, W_4, W_3, W_2, W_1, W_0

2. W_5 conditions the TRUE FALSE selection on the vectors.

3. W_4 and W_3 condition the U, L selection as:

W_4	W_3	Selection
0	0	U
0	1	L
1	1	LU
1	0	UL

4. W_2, W_1 and W_0 condition the vector selection as follows:

W_2	W_1	W_0	Selection
0	0	0	S
0	0	0	D
0	1	1	E
0	1	0	SE
1	1	0	SD
1	1	1	DE
1	0	1	ED

TABLE 2-9. BB CONTROL

BB	Effect
00	Write Zero
01	Write One
11	Invert
10	Retain

2.6 TIMING OF ASSOCIATIVE MEMORY CONFIGURATION

2.6.1 Introduction

This paragraph discusses and develops the elements involved in instruction timing for the GAP, A1, A2, and A3 configurations. Timing information used in the evaluation of these configurations is also presented. The initial subparagraphs of this paragraph are concerned with the basic elements which comprise the overall timings, their interactions, and some of the problems of deriving or developing an overall set of timings for evaluation purposes.

The greatest timing problems are presented by the Multiple Pass Searches (maximum, minimum, next higher than, next lower than) present in the GAP and A3 configurations. The complexity of timing multiple pass searches is due to the search algorithms: that is, the search time is a function of the actual data search and may vary widely for different sets of data. GAP timing is complicated because of the configuration of the GAP 1604-B system. In this system, GAP is used as a peripheral device, executing its own program from 1604-B memory. In use, GAP must be synchronized with the 1604-B so that the 1604-B program may obtain and use the results of searches performed in the GAP. However, GAP and 1604-B are timed independently and run independently, thereby resulting in complicated timing situations involving both the load on the 1604-B and the elapsed time for GAP to complete a given operation sequence.

In the A1, A2, and A3 configurations, timing is much more straightforward. A3 timing is basically the same as 1604-B timing because the associative memory is fully integrated into the 1604-B. The major additional problem presented by A3 is the timing of the multiple task search operations. In A1 and A2, most instructions are executed within the EXF time of the 1604-B. Thus, for A1 and A2, the EXF, INT, and OUT instructions control the timing of both the 1604-B and the associative memories. Timing for these operations is found in the programming manual for the 1604-B.* The logic capability of the A2 configuration presents one minor complexity that is treated in subsequent paragraphs.

The following paragraphs develop the elements which comprise overall timing numbers for the configurations studied.

* Control Data 1604 Computer Programming Manual. CDC Publication No. 167b, Rev. August, 1962.

2.6.2 Basic Timing Elements

In the use of the GAP, A1, A2, and A3 configurations, there are three basic elements which contribute to the execution time of instructions:

(1) Memory accessing

(2) Resolution of responses

(3) Instruction algorithms

2.6.2.1 Memory Accessing. The 1604-B employs two independent (and independently timed) banks of memory. Each bank has a read access time of 2.2 microseconds and a full cycle time of 6.4 microseconds. CDC claims an effective memory time of 4.8 microseconds, but includes in this time the effective overlap obtained by the two instructions per word format of the 1604-B. Within the 1604-B, several mechanisms compete for memory access. These mechanisms are the program, the I/O system, and certain internal machine features (real-time clock, interrupt system, etc.). Each I/O channel (including the DMA channel of the GAP*) is polled, in a fixed sequence, by an I/O scanner to determine when memory access must be granted to the I/O system and which channel will obtain access.

Two separate times are needed to describe the effect of I/O operation on system timing. The first time represents the amount of available 1604-B memory time consumed by the I/O transfer; the second time represents the elapsed time required to satisfy the I/O request for memory access.

(1) Load on the 1604-B. The 1604-B time consumed by I/O operation is a function of how much memory overlap is obtained in the overall sequence of memory accesses by all using mechanisms. With perfect overlap, the effective memory rate is 4.8 microseconds per access; with no overlap, the memory rate is 6.4 microseconds per access. In the present circumstances it is reasonable to split the difference and, by exercising engineering judgment, adopt an I/O load on the 1604-B of 5.5 microseconds per access. Using this loading number means that, for timing and load analysis purposes, each word transferred in or out via the 1604-B I/O system (which includes the DMA channel) requires 11 microseconds of 1604-B time for normal I/O and 5.5 microseconds of 1604-B time for I/O on the DMA channel (in buffer mode**).

* The DMA channel in burst mode is a special case, treated separately.

** Each I/O transfer in the 1604-B requires one memory access for the I/O control word and one access for the word transferred, except the DMA channel which does not require the control word access.

In the A3 configuration, the AM is logically a part of the 1604-B memory structure. In this configuration the 1604-B has a third bank of memory, independent and independently timed with a 4-microsecond full cycle time.

In normal operation, this third logical and physical memory bank has no effect. But, because the AM is composed of contiguously numbered memory locations (the last 2,048 words), the kinds of accessing overlaps obtained are more easily analyzed when they involve the AM. The benefits gained should be obvious. For the present purposes, they may be ignored except for the case of I/O transfers to or from AM. In this case, each I/O word transferred requires only 4 microseconds of 1604-B time so long as a program is not being executed from the AM. For other cases, including 1604-B instruction timing, it is assumed that the AM cycle is the same as the 1604-B cycle. These assumptions introduce additional timing complexity only when there are clear, significant, and easy-to-evaluate benefits.

(2) Elapsed Time. The elapsed time for the AM operations is widely variable and extremely complex to evaluate. It is composed of several parts, each of which is variable, and their combined effects are difficult to evaluate. The problems involved are most difficult in the case of GAP; for A1, A2, and A3, timing is much more straightforward and simple. A general sequence of the events which are involved in instruction execution is shown in Figure 2-23.

In the preceding paragraphs the value 5.5-microseconds has been assigned to access operations (such as t_1-t_2 and t_3-t_4) to 1604-B memory. The times represented by the intervals t_0-t_1, t_2-t_3, etc., are a function of both the level of I/O activity of the 1604-B and of the instruction sequence being executed by the 1604-B. The access delay interval may be any value from 0.0 to 77 microseconds in the 1604-B system. The first value (0.0) could occur under any circumstances, but most likely when no I/O or interrupt activity is present. The second value (77) can only occur when all I/O channels are active.

The major complexities arise in the GAP configuration because the 1604-B and the GAP are running independently and must frequently be resynchronized. The GAP is competing with the 1604-B program (and I/O) for access to memory. In A1, A2, and A3 there is no such competition, except in very special cases which are treated separately. Our major concern here is to develop values for the elapsed time for each GAP instruction. The elapsed time of a sequence of 1604-B instructions* is easily derived from a consideration

* In the sense used here the 1604-B instructions subsume the associative features of A3. Elapsed time of a sequence is the sum of instruction times plus 11 microseconds per word transferred during the execution of the sequence. For A1 and A2, elapsed time is simply instruction time on the transfer channel.

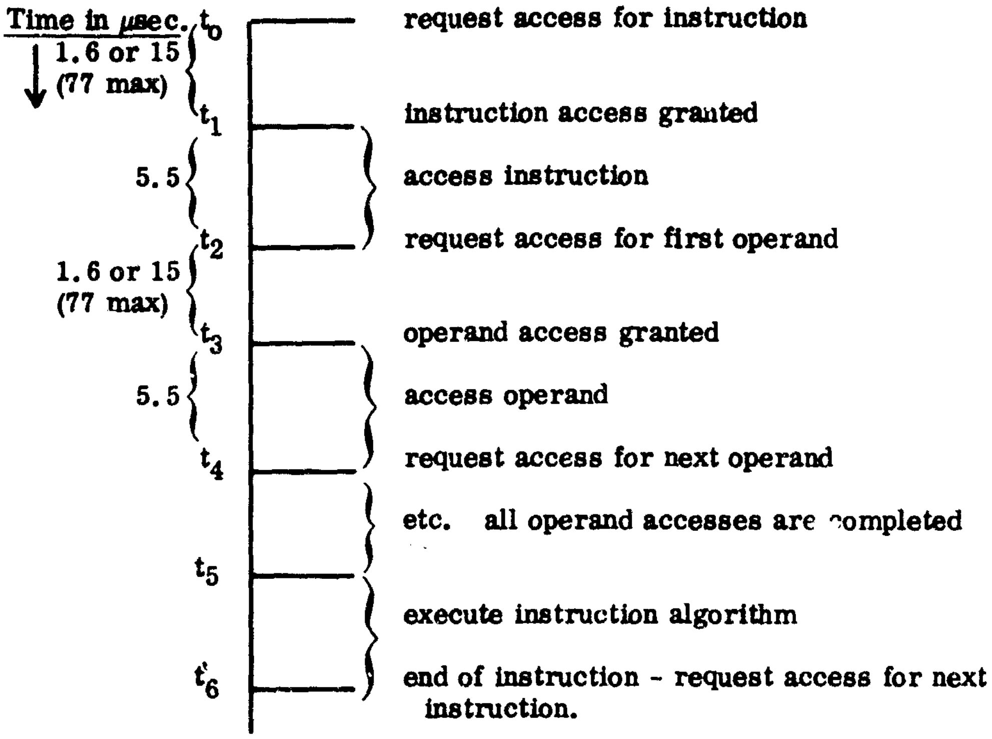

Time in μsec.
t_0
request access for instruction
1.6 or 15
(77 max)
t_1
instruction access granted
5.5
access instruction
t_2
request access for first operand
1.6 or 15
(77 max)
t_3
operand access granted
5.5
access operand
t_4
request access for next operand
etc. all operand accesses are completed
t_5
execute instruction algorithm
t_6'
end of instruction - request access for next instruction.

Figure 2-23. Instruction Execution Sequence

of the instruction times and the I/O (including DMA) load at the time of execution of the sequence.

Before the GAP can gain access to 1604-B memory, its request must be recognized by the I/O scanner. The delay till access is granted due to two factors: (1) the time required for the scanner to recognize the request, and (2) the time required for the request to fit into the current interlace pattern in 1604-B memory. Provision for the latter factor has been made in the assumption of a 5.5-microsecond access rate to 1604-B memory. The scanner time is a function of I/O activity. The scanner inspects all I/O channels in 3.2 microseconds if none are requesting service.* Thus, for the case of no I/O operation, 1.6 microseconds may be used as an average access delay. In the worst case a 77-microsecond delay could be obtained with all I/O channels active. For evaluation purposes, 15 microseconds has been selected as a representative delay value for moderate I/O activity.** The two values, 1.6 microseconds and 15 microseconds, will be used in timing evaluations to represent the memory access delay per word experienced by the GAP. Remember that this delay could be as high as 77 microseconds per word.

The final element of elapsed time is the time required by the AM algorithms to execute the instructions after all operands are fetched from 1604-B memory. These times are developed in the following paragraphs.

2.6.2.2 Response Resolution. Because the resolution and/or counting of responders is involved in the mechanization of many of the GAP and A3 (to a lesser extent in A1 and A2) instructions, the response resolver (RR) will be described before discussing the instructions. We have very little direct information on the implementation of the RR. However, based upon certain clues and claims in the Goodyear literature which has been made available to us, we have postulated a probable implementation. Our implementation is consistent with the known facts about the GAP resolver, the requirement placed on the RR by the Goodyear claims, and good engineering practice within the milieu of the GAP design.

(1) Resolver Logic. The response resolver's task is to find the next, or first, bit in the S register which is a ONE. It is also used to count the number of ONEs in the S register. We assume a logical structure as shown in Figure 2-24.

* See CDC publication #60110100, July 1964, I/O Specification for the CDC 1604-B Computer.

** Moderate I/O activity implies one high-speed device and one low-speed device in operation.

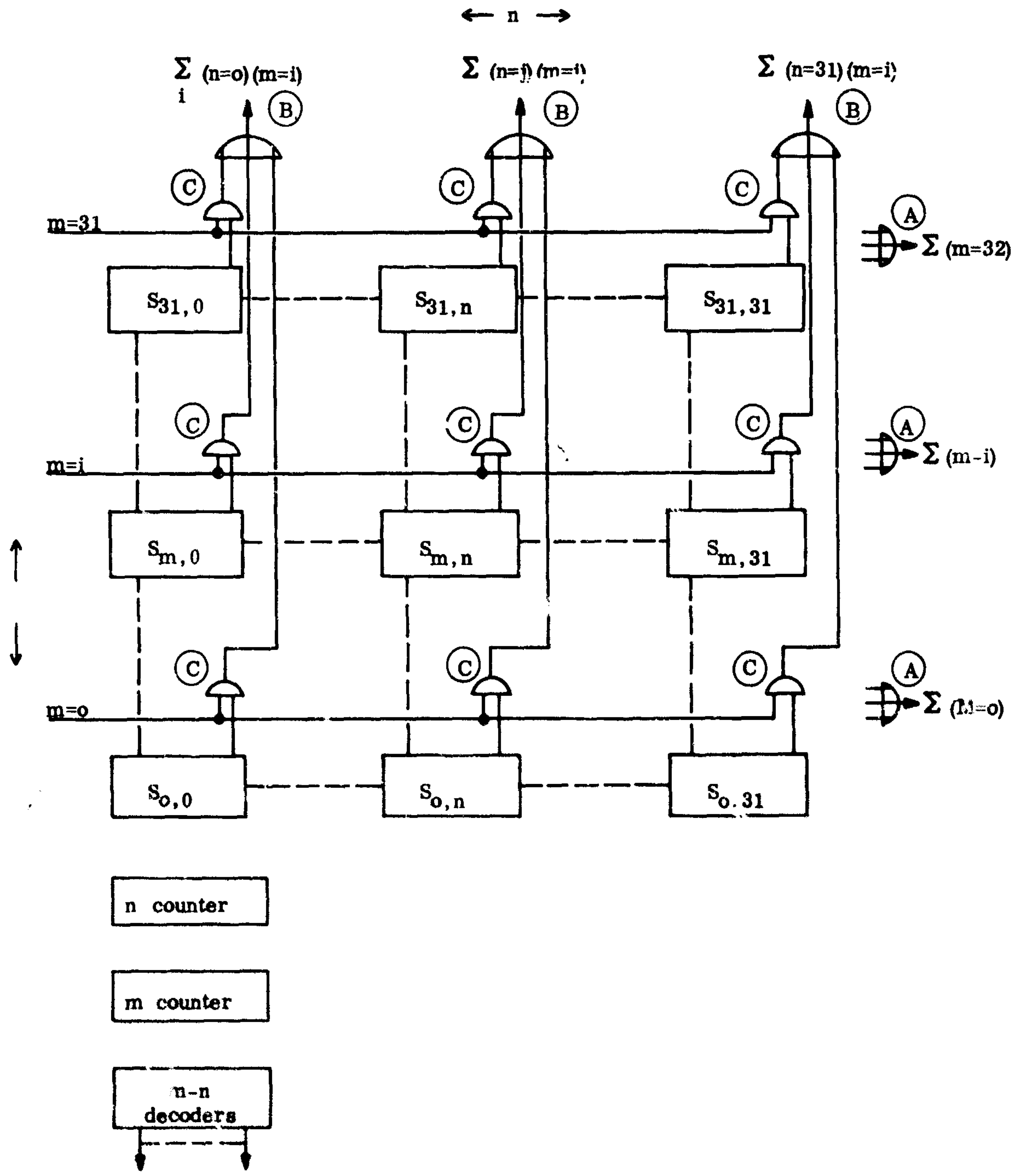


Figure 2-24. Skeleton of Response Resolver

The RR network is developed by considering the 1,024 bits of S to be arranged into a 32x32 matrix. The matrix position of a bit of S is then given, uniquely, by a pair of five-bit(or base 32) subscripts, m and n. In Figure 2-24 the matrix is shown with m rows and n columns and the subscripts are shown on the representative bits of S.

The resolution procedure is to scan the rows of the matrix to determine if there is a ONE in a given row. If a row is found to contain a ONE, the columns of that row are scanned to determine the exact position of the ONE. The subscripts of the S position in the matrix then yield a 10-bit number which is the index of the ONE that has been located. Because of the one-to-one mapping of AM words into the bits of S, the index on S is also the low order 10 bits of the address of the word. Thus,a two-dimensional scanning scheme to resolve responses is employed.

The implementation of this scheme is indicated in Figure 2-24. The gates marked Ⓐ serve to OR together the ONEs of each row of S. They are 32-input gates and there are 32 of them. The outputs of the 32 gates are scanned by the m counter and its decoder. The gates marked Ⓑ are identical to the Ⓐ gates, except that they OR together the gated columns. The Ⓒ gates are enabled by the m decoder to allow the selected row to appear at the Ⓑ gates. The Ⓑ outputs are then scanned by the n counter and its decoder. When a ONE is found, the m and n counters contain the 10-bit index of the position in S of the ONE. Once a scan of a row has been started, it must be completed before stepping to the next row to ensure that all the ONEs in the row are resolved.

(2) Resolver Timing. Goodyear Aerospace Corporation claims that the worst case resolution time is one microsecond. In this implementation the worst case would be a single responder at $S_{(31)}$ (31) and it would require 64 scan steps to locate it. Thus, each scan step must take about 16 nanoseconds.

The worst case to count the number of ONEs in S would be to have a ONE in each row of S, thus requiring 1,024 scan steps or about 16 microseconds.

A much more difficult problem is to decide the resolution time for representative cases of responders. Clearly, this is a complicated function of the data pattern in memory. Appendix B attempts to develop resolution time as a function of the probability of a given bit of S being ONE, with the result shown in Figure 2-25.

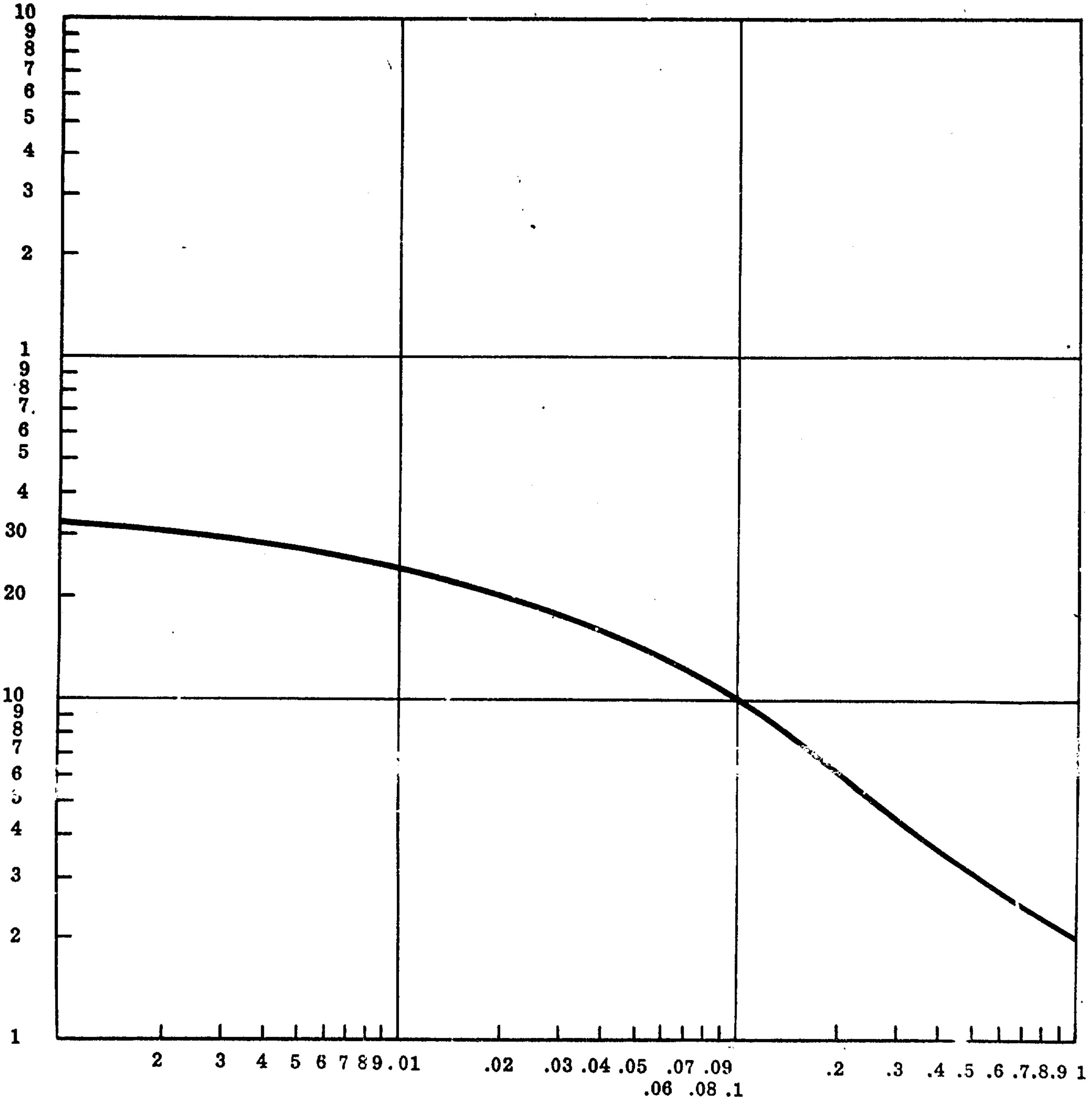


Figure 2-25. Resolution Steps vs. Probability of ONE in Given Position of S

2.6.2.3 Instruction Algorithms. The timings presented in this paragraph are based on our knowledge of the actual GAP implementation, the response resolver described above, and our engineering judgment. Many assumptions have been made to develop these times; most are unimportant. The critical assumptions are:

(1) In AM search operations only those columns are interrogated whose mask bit is ONE. All other columns are skipped. An interrogation requires 0.1 microsecond; a skip requires 0.05 microsecond.

(2) Search operations are terminated as soon as it is known that a result has been developed.

(3) The response resolver, and its timing, is as previously described.

(4) Timings are developed for searching 1,024 words, either the upper or lower half of AM. The results are directly extendable to searches of the full memory.

This paragraph is concerned only with the search instructions; the other AM instructions are standard in any computer. No description of the AM algorithms beyond the point necessary to develop their timing is given since this material can be obtained by a study of the literature.

Of the 11 GAP search operations, six (EMC, MMC, LTC, LEC, GTC, GEC) are accomplished in a single pass* through the memory, a seventh (BLC) requires two passes, and the remaining four a variable number of variable length passes.

(1) Single-Pass Searches. The timing of the single-pass searches is a function of the number of bit positions searched. If K is the number of positions whose mask bit is ONE, a pass through memory requires $T_1 = 0.1K + 0.05\ (48-K) + 0.2$ microseconds for the GAP, $T_1 = 0.1K + 0.05\ (49-K) + 0.1$ microseconds for A1 and A3 and $T_1 = 0.1K + 0.05\ (57-K) + 0.2$ microseconds for A2. The maximum search is thus _5 microseconds. If it is assumed that a representative search would inspect 24 of 48 bits, a representative search time of 3.8 microseconds per single pass search per half memory (1,024 words) can be used.

* A pass implies the ordered interrogation of each bit column of the AM.

The two-pass search (BLC) is simply two single-pass searches executed in series. Thus, the maximum time is [illegible] microseconds and a representative time is 7.6 microseconds.

(2) Multiple-Pass Searches. The multiple-pass searches are MAX, MIN, NLC, and NHC. The latter two are simply the former two preceded by a single-pass search (LEC or GEC). Some of the searches in A3 are the same as in the GAP as shown below.

MXF = MAX
MNF = MIN
NHF = NHC
NLF = NLC

Execution of these searches requires the use of the S register and one other (or "buffer") register (for each half of memory searched). The MAX and MIN searches require the counting of S (at least up to a count of 2) after each interrogation of a column. Thus, a MAX or MIN search would require:

$$T_2 = 0.1K + 0.05\ (48-K) + 0.2 + 0.016\ (49-K)$$

$$\left[34 + \sum_{i=1}^{K} (m_2 + n_2)\right] \mu\text{sec.}$$

In this expression, m_2 and n_2 are the subscripts of the S position of the second responder in S at the completion of each interrogation. A worst case of MAX or MIN would thus require:

$$T_2 = 0.1 \times 48 + 0.2 + 0.016\ (49)\ \left[34 + 64\right] \mu\text{sec.}$$

$$= 4.8 + 0.2 + 76.8\ \mu\text{sec.}$$

$$= 81.8\ \mu\text{sec.}$$

A worst case of NHC and NLC is simply a worst case MAX or MIN plus 5 microseconds or 86.8 microseconds. (See Appendix B for detailed timing equations.)

(3) A3 Slow Searches. In the MAX and MIN GAP searches, the second 1,024-bit register required is used to hold intermediate results because a column interrogation could produce all ZEROs in S. On the slow A3 equivalent searches, when all ZEROs appear in S, the column causing the ZERO result is masked so that it may be skipped in the future and the entire procedure is restarted. While in the worst case this takes much longer than the GAP searches, the disparity between the times for representative data is not so great. The A3 searches do not destroy the contents of another 1,024-bit register as do the GAP searches. Appendix B develops the worst case and representative case times for the A3 slow searches.

2.6.3 Instruction Times

Using the elemental instruction step timing development in the preceding paragraphs, this paragraph combines the needed elements for each instruction of the GAP, A1, A2, and A3 configurations to give summary tables of instruction times.

2.6.3.1 GAP Instruction Times. Table 2-10 shows the times necessary to develop running time estimates for the GAP configuration. Appendix A lists the GAP instructions.

The first column of the table shows the average 1604-B memory time consumed by each GAP instruction. The second column shows the elapsed times to be expected under the conditions of no I/O activity and moderate I/O activity associated with the 1604-B. Under conditions of heavy I/O activity these times could be much larger. The last column shows the representative, or fixed, AM operation time and the maximum AM time which might be required. The average elapsed time figures employ the representative times from column 3. Again, the elapsed times could be much larger if the maximum AM times are used.

A certain amount of overlap of 1604-B and GAP operation has been allowed in developing the elapsed time numbers. Using the information presented, timing numbers can be developed for any critical 1604-B or AM operations which do not fit the environment presented here. It is suggested that in most cases the numbers presented are sufficient. Times for arithmetic operations in the GAP have been assumed.

Once a burst mode operation is started in the GAP, each word transferred requires 4.8 microseconds of elapsed and 1604-B time. Note, however, that the burst mode is as yet undefined by Goodyear.

2.6.3.2 A1 Instruction Times. A1 instruction times are much simpler to determine than the GAP instruction times. Two kinds of operations are performed with the A1. First, EXF codes are sent from the 1604-B to the A1 to specify what instruction is to be performed. In all cases, A1 responds within the 6 to 8-microsecond EXF time of the 1604-B. Thus, all EXF codes and the corresponding actions in A1 are accomplished in 1604 time. Second, A1 responds to INT and OUT instructions from the 1604-B. Again, any A1 action required is completed in the transfer time of the 1604-B. (This time is found in the 1604-B manual.)

TABLE 2-10. GAP INSTRUCTION TIMES
(for 1024 words searched)

Instruction	Average 1604-B Time Used (micro- seconds)	Average Elapsed Time (microseconds)		AM Operation Time (microseconds)	
		NO I/O	MODER- ATE I/O	Representa- tive Average	Maximum
RESUME	6-8	6-8			
FORCE	6-8	6-8			
SENSE	6-8	6-8			
CLEAR	6-8	6-8			
BLC	22.0	43.1	89.6	7.6	10.0
CPX	16.5	note 1	note 1	note 1	note 1
DEL	5.5	10.1	23.5	3	
EAR	5.5	7.1+4.1/wd	20.5+4.1/wd	4.1/wd	
EFR	5.5	9.6	24.6	4.1	
EMC	16.5	32.2	64.3	3.8	5.0
EMY	5.5	27.1	40.5	2.0	
END	note 1	note 1	note 1	note 1	note 1
GEC	16.5	32.2	64.3	3.8	5.0
GTC	16.5	32.2	64.3	3.8	5.0
HLT	5.5	7.1	20.5	Nil	
ICI	5.5	10.1	23.5	3	
JIH	5.5	10.1	23.5	3	
JIL	5.5	10.1	23.5	3	
JNR	5.5	7.1	20.5	Nil	1.0
JUC	5.5	7.1	20.5	Nil	
LDI	11-16.5 note 2	14.2-28.4 note 2	41-60.5 note 2	Nil	
LDR	5.5	8.1	21.5	1	
LEC	16.5	32.2	64.3	3.8	5.0
LTC	16.5	32.2	64.3	3.8	5.0
MAX	11.0	20.3	47.1	6.1	81.8
MIN	11.0	20.3	47.1	6.1	81.8
MMC	16.5	32.2	64.3	3.8	5.0
NHC	16.5	39.5	71.6	11.1	86.8
NLC	16.5	39.5	71.6	11.1	86.8
NOP	5.5	7.1	20.5	Nil	
RAF	11.0	15.3	42.1	1.1	2
RBE	5.5	7.1+4/wd	20.5+4/wd	4/wd	
RBL note 3	11+5.5/wd	14.2+7.1/wd	41+20.5/wd	4/wd	
RBU note 3	5.5+5.5/wd	7.1+7.1/wd	20.5+20.5/wd	4/wd	
RCF	11.0	18.3	45.1	4.1	5.0
RCR	11.0	15.2	42.0	1.0	16.0
RDA note 3	5.5+5.5/wd	7.1+7.1/wd	20.5+20.5/wd	1/wd	1/wd
RDC note 3	5.5+5.5/wd	11.1+7.1/wd	24.5+20.5/wd	4/wd	5/wd

TABLE 2-10. GAP INSTRUCTION TIMES (Cont.)

Instructions	Average 1604-B Time-Used (micro-seconds)	Average Elapsed Time (microseconds)		AM Operation Time (microseconds)	
		NO I/O	MODER-ATE I/O	Representa-tive Average	Maximum
RUM note 3	5.5+5.5/wd	11.1+7.1/wd	24.5+20.5/wd	4/wd	
SCF	note 1	note 1	note 1	note 1	note 1
SIX	11-16.5	14.2-28.4	41-60.5		
	note 2	note 2	note 2	Nil	
SPJ	11-16.5	14.2-28.4	41-60.5		
	note 2	note 2	note 2	Nil	
WCR	16.5	28.4+4/wd	60.5+4/wd	4/wd	
WFA note 3	11+5.5/wd	14.2+7.1/wd	41+20.5/wd	.1+4/wd	
WIR note 3	11+5.5/wd	14.2+7.1/wd	41+20.5/wd	4/wd	

Notes:

1. The complex search must be evaluated in terms of the individual parts of the search sequence.
2. Number pair depends on the use of indirect addressing.
3. In buffer mode.

A single exception exists for A1 (i.e., the counting of responders). As discussed previously, the maximum time to count responders is 16 microseconds. Thus, the 1604-B INT instruction following an RCS instruction to A1 may be delayed up to 16 microseconds.

2.6.3.3 A2 Instruction Times. A2 responds to three kinds of commands:

(1) Instructions delivered as EXF codes.

(2) Instructions executed from the I register.

(3) I/O transfer operations on Channel 7.

All A2 instructions delivered as EXF codes are completed by A2 within the 1604-B EXF time of 6-8 microseconds (depending on the location, upper or lower, of the 1604-B EXF instruction). In any event the 1604-B is free to execute instructions, but A2 will ignore instructions to it if it is still occupied with a previous operation. The TRY or interrupt facilities may be used to determine when A2 is ready to accept a new operation. All I/O operations with A2 proceed at the maximum 1604-B speed as listed in the 1604-B manual.

The times listed in Table 2-11 are the times A2 will require to complete operations in cores in which the operation times are greater than the 6-8 microsecond EXF time.

TABLE 2-11. A2 TIMINGS

Instruction	Time (μsec.)	Notes
RCS	6-102	average 50
RCS	6-9.6	
XLI	6-8 min.	2 plus time for the 8 bit instruction
LO	6-8 or 160	6-8 for FØ, F3, F5, F10, F12, F15; 160 all other
SO	4096	
SH	3¼	

2.6.3.4 A3 Instruction Times. Timing the A3 configuration is considerably simpler than timing the GAP configuration. Operations in A3 are performed as in any normal general-purpose computer. There is no competition between AM and 1604-B programs for memory access since they are one and the same thing. The only time the AM runs independently of the 1604-B is during a WCR instruction. Thus, subject to AM timing variations and effective memory overlap of instructions and data in 1604-B memory, the A3 instruction times are directly useable in timing of programming sequences.

There are five classes of instructions in A3 as listed below:

(1) Normal 1604-B Instructions
(2) 1604-B-Like Instructions
(3) Search Instructions
(4) Control Instructions
(5) Miscellaneous Instructions

The timing of each class of instructions is treated separately in the following paragraphs.

(1) Normal 1604-B Instructions. The timing of normal 1604-B instructions is exactly as specified in the 1604-B manual.

(2) 1604-B-Like Instructions. The execution times for the entire set of 1604-B-like instructions are the same as the corresponding 1604-B instructions plus the time needed to resolve an operand. For cases in which there are a reasonably large number of ONEs in the register being resolved, resolution time is so small that it may be assumed to be absorbed by the faster cycle of AM memory as compared with 1604-B memory. In the worst case, the resolution time is one microsecond. For timing purposes, the 1604-B-like instructions require the same time as their normal 1604-B counterparts.

(3) Search Instructions. The timing of the A3 search instructions is given by Table 2-12. The entries in the table give the average and maximum times for the instructions including all memory accesses for the instruction and any necessary data.

TABLE 2-12. A3 SEARCH INSTRUCTION TIMING

	Time in Microseconds	
INSTRUCTION	AVERAGE	MAXIMUM
EMC	6.2	7.4
MMC	6.2	7.4
GEC	6.2	7.4
GTC	6.2	7.4
LEC	6.2	7.4
LTC	6.2	7.4
MXF	8.5	84.2
MXS	10.9	982.4
MNF	8.5	84.2

TABLE 2-12. A3 SEARCH INSTRUCTION TIMING (Cont.,

INSTRUCTION	Time in Microseconds AVERAGE	MAXIMUM
MNS	10.9	982.4
NHF	13.5	89.2
NHS	31.4	1102.4
NLF	13.5	89.2
NLS	31.4	1102.4
NOP	3.0	3.0
EXT	3.0	3.0

The timing of these operations, or more exactly the associative part of them, was developed in the preceding paragraphs. An instruction fetch time of 2.4 microseconds was used. (This is the same as the time CDC uses in their instruction timing.)

(4) Control Instructions. The timing of the control instructions is given in Table 2-13. The times include all necessary instructions and operand accesses.

TABLE 2-13. A3 CONTROL INSTRUCTION TIMING

INSTRUCTION	TIMING
WCR	3.0 Note 1
RCA	3.1 Note 2
RCQ	3.1 Note 2
LMA	3.0
LMQ	3.0
LAM	3.0
LQM	3.0
LCA	3.0
LCQ	3.0
LAC	3.0
LQC	3.0
LSA	3.0
LSC	3.0
LAS	3.0
LQS	3.0
LRA	3.0
LRQ	3.0
LAR	3.0
LQR	3.0

Note: 1. After the 2 microseconds the 1604-B continues to execute instructions, but the AM is busy for 4 microseconds per word.

Note: (contd.)

2. Assumes a resolver counting time of 0.1 microsecond. The maximum time is 16 microseconds so that maximum instruction time is 19 microseconds.

(5) Miscellaneous Instructions. The A3 miscellaneous instructions are all EXF instructions and require 6 to 8 microseconds.

2.7 COST

2.7.1 Introduction

The cost figures for the four hybrid associative memory configurations were developed as a basis for cost performance comparisons. The memories are: Goodyear's as defined by their Proposal GAP-2549 and their programming manual, and AUERBACH's A1, A2, and A3. Since the cost figures are developed as a basis for comparison rather than as price projections for budget or other purposes, their relative values are more significant and probably more accurate than their absolute values.

The fundamental assumption made in the design of the configurations considered here was that they each must use present AM technology. This is further interpreted to mean that the 2,048-word Goodyear BILOC array would be used for the two memories designed by AUERBACH as well as for the Goodyear version because:

(1) Its cost appeared to be similar to that of other similar constructions such as UNIVAC's BICORE, Raytheon's BIAX, andothers. This does not mean that the BILOC represents the lowest cost either now or in the future, particularly when compared with other approaches which use a different AM organization such as delay lines, or perhaps thin films.

(2) Use of the identical array shows clearly the relative cost of the remaining parts of the memory and consequently the effects of adding or removing particular features.

(3) Since RADC has ordered a BILOC memory, the results of our study may be directly applied to the RADC situation.

(4) Ground rule requires it (see Paragraph 2.2).

Since a memory is actually being built, it is assumed that R&D costs will not be borne directly by future models. Accordingly, technique development costs are excluded from the cost figures below. Design costs and memory development cost are also excluded because the per-unit development cost will vary depending on production volume. The resulting cost figures then are indicative of hardware production costs including overhead and profit.

2.7.2 Cost of the Goodyear Associative Memory

The cost estimate of the Goodyear memory is divided into the following four categories:

(1) Response store
(2) Memory array and associated electronics
(3) Data registers, control, and other electronics
(4) Memory design and development

Each of these is discussed in the following paragraphs.

2.7.2.1 The Response Store. The response store cost was estimated by first designing one which is thought to be close to that used by Goodyear. Next, the number of circuit elements necessary to implement that design was estimated. And last, the cost of the elements and the various costs of assembling them into a response store were estimated. The design of the response store is shown in Figure 2-26; the dotted lines enclose circuit elements of a Fairchild Semiconductor family of industrial integrated circuits.

The numbers within the dotted areas refer to the first three digits of the Fairchild catalog number. Table 2-14 shows how the parts list was converted into an overall response store cost per stage. When the per stage cost is multiplied by the 1,024 stages, the resulting response store cost is about $152,300.

2.7.2.2 Memory Array. The cost of the BILOC array and associated electronics was estimated by individually estimating costs of the component parts and totaling them. The parts as listed in Table 2-15 were taken from the Goodyear Sketch, SK 330N000-001, dated 3-31-65. Table 2-15 also shows the estimated cost including overhead and profit of assembling the array and associated electronics — the total is $144,160. Note that this cost will be invariant among the three memories.

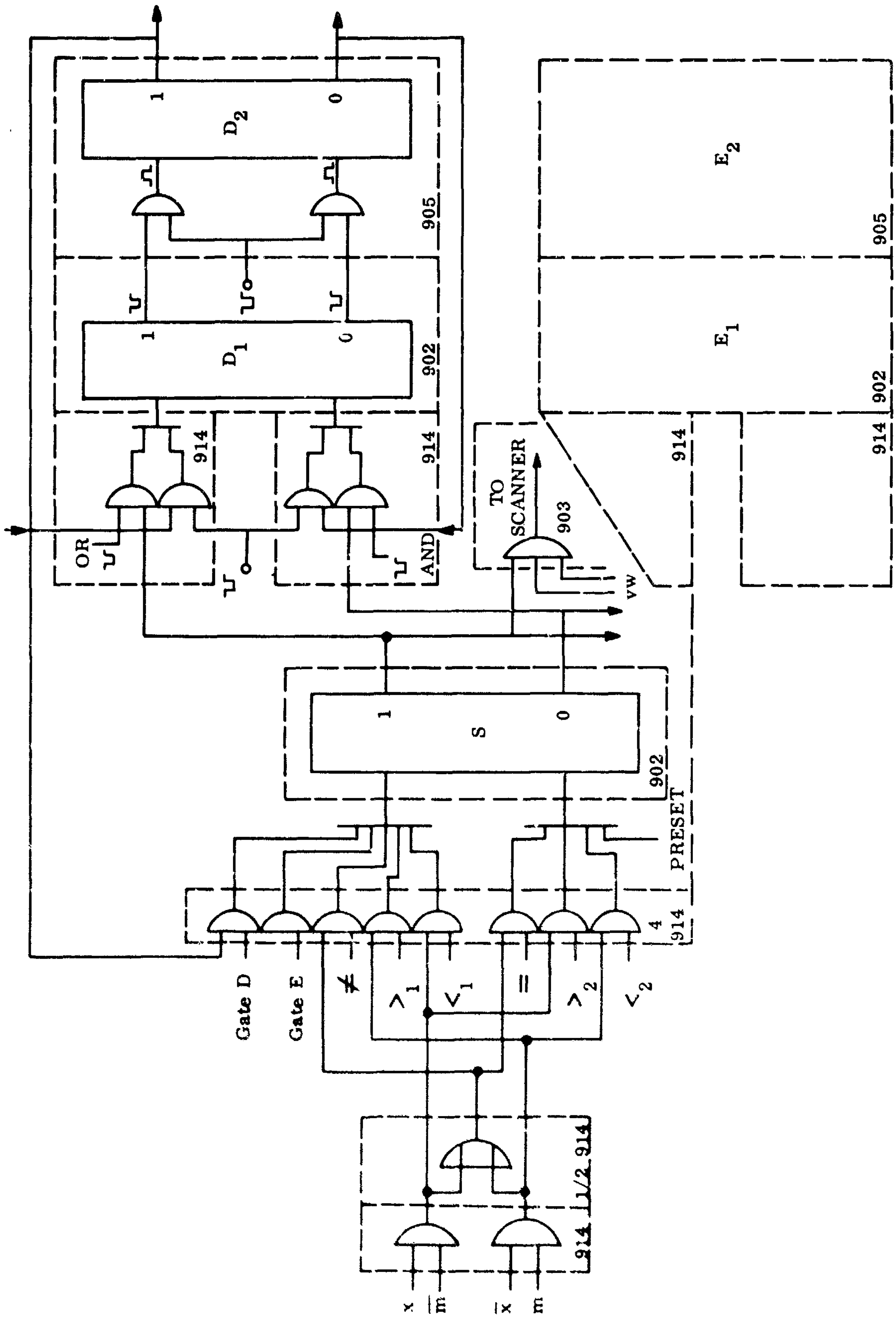


Figure 2-26. Estimated Goodyear Response Store Configuration

TABLE 2-15. ESTIMATED COSTS OF ARRAY AND ASSOCIATED ELECTRONICS

Item	Number	Estimated Cost
Memory Array	1	$20,000
Non-Destructive Sense Amps (@ $90 amp)	1024	92,160
Write Switches	100	5,000
Read Amplifiers	50	5,000
Address Decoding Matrix	1	3,000
Read/Write Drivers	65	4,000
Busy Bit Reset Driver	1	500
1/3 Write Drivers	50	4,500
Interrogate Drivers	98	10,000
TOTAL		$144,160

TABLE 2-16. COST ESTIMATE FOR REGISTERS AND OTHER CIRCUITS

Category		Estimated Cost
Registers		$10,120
Counters		5,000
Other		
Line Terminations	200	
Parity Generator	500	
Selection Gates	300	
Line Drivers	500	
Response Resolver	5,000	
Test Flip-Flop	100	
Instruction Decoder	1,000	
Index Adder	3,000	
Sub-Total		10,600
Control		14,280
TOTAL		$40,000

2.7.2.3 Data Registers, Control, and Other Electronics. The cost of these items was determined by dividing them into the four categories:

(1) Registers
(2) Counters
(3) Other Circuits
(4) Control

From the Goodyear Sketch, SK 330N000-001, one can see that there are about 250 register stages, about 50 counter stages, and other circuits as listed in Table 2-16. Costs for each of the four categories are shown in the table along with an estimate for control circuitry such that the total for the category is about $40,000.

TABLE 2-14. GOODYEAR RESPONSE STORE ESTIMATE

Calculation of Integrated Circuit Cost

Number of ICs per Stage	Type	Description	Quantity Price Per Unit	Total
2	905	flip-flop	$4.35	$ 8.70
3	902	flip-flop	5.50	16.50
9-1/2	914	gates	2.65	25.38
1	903	gate	2.55	2.55
		TOTAL		$53.13

Calculation of Total Cost

Item	Cost
Integrated Circuits	$ 53.13
Etched Circuit Card and Connector	12.00
Card Assembly	5.00
Rack Assembly	20.00
Sub Total	$ 90.13
Overhead Factor (x1.5) Yields	135.20
Profit (+10%)	13.52
TOTAL	$148.72 per stage

2.7.2.4 Memory Design and Development. A total of $170,000 is estimated to represent several man-years at various skill levels. This brings the total estimated cost of the present Goodyear memory (without the M1218 interface) to $506,500 as shown in Table 2-17. The table also presents the partial total of $336,500 by excluding the development costs. This figure will be used for comparison with the other memories. The percentage costs are also given to show more clearly how the costs are divided.

TABLE 2-17. ESTIMATED COSTS FOR GAP
(including integration circuits but not including cost to modify the 1604-B)

	Present Memory	Subsequent Copies	Percentage of Cost for Subsequent Copies
Response Store	152.3	152.3	45
Array and Associated Electronics	144.2	144.2	43
Registers and Logic	40.0	40	12
Logic Design	80.0		
Production Design	90.0		
Sub-Total	506.5	336.5	100

(Assume profit included)

2.7.3 Cost of A1

The cost of the A1 model is different from the cost of GAP in two respects. First, the much simpler response store shown in Figure 2-27 has been assumed. Second, fewer registers and control circuits are used, thereby reducing the design costs as well as the cost of the logic circuitry. The array is assumed the same as in the Goodyear and therefore costs the same.

The cost of the A1 response store is estimated from the fact that one type 902 flip-flop, three type 914 gates, and one type 903 gate modules are needed per stage. Table 2-18 shows the calculation of circuit cost and the conversion of this number into a total cost of $44.05 per stage.

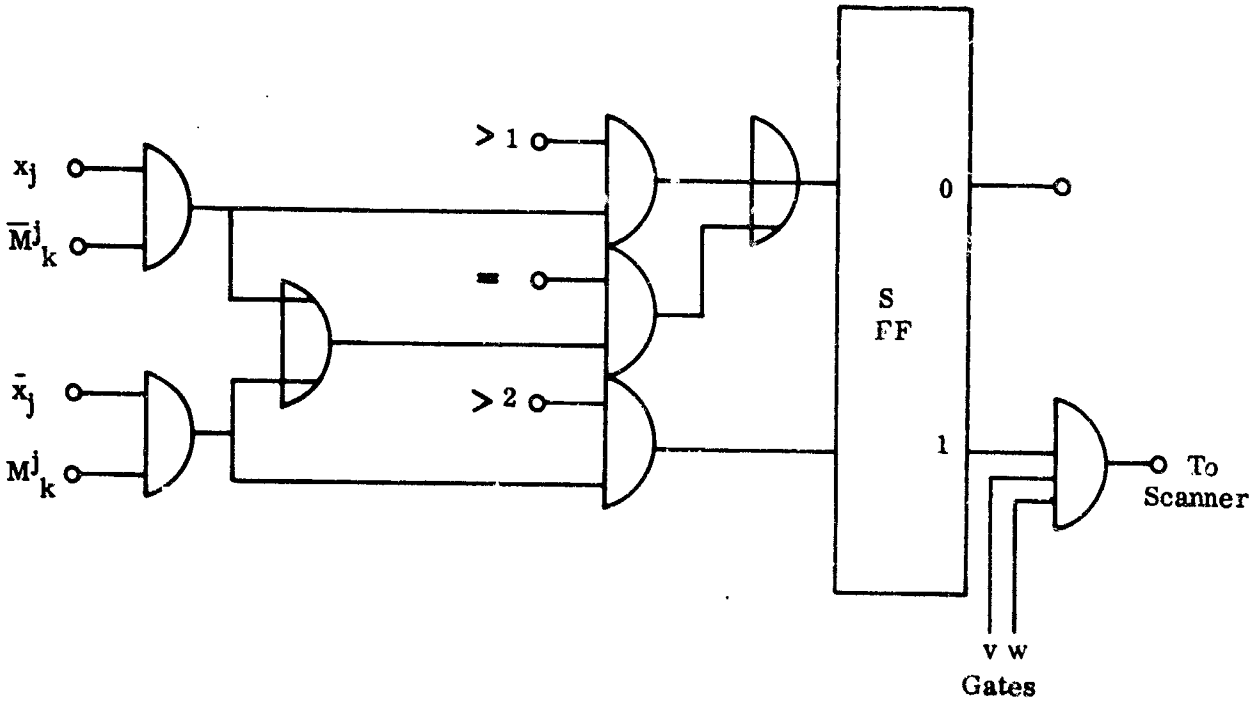


Figure 2-27. A1 Response Store

TABLE 2-18. A1 Response Store Costs

Circuit Costs for A1 Response Store

Number Required	Fairchild Type	Description	Quantity Price Per Unit	Total
1	902	flip-flop	$5.50	$ 5.50
3	914	gate	2.65	7.95
1	903	gate	2.55	2.55
			TOTAL	$16.00

Total Response Store Cost Per Stage

Cost Element	Cost
Integrated Circuits	$16.00
Card and Connector	4.00
Card Assembly	1.70
Rack Assembly	5.00
Sub-Total	$26.70
Overhead Factor (1.5) Yields	40.05
Profit (10%)	4.00
Total	$44.05 per stage

2.7.4 Cost of A2

The difference between A1 and A2 is primarily in part of the instruction set and in the addition of a 32-bit supplement to the response store as indicated by Figure 2- It is estimated that each of these two changes would add approximately $5,000 so that its total cost would be $214,000.

2.7.5 Cost of A3

The difference between the cost of A3 and GAP is primarily in that part of the system which interfaces with the 1604. Because the A3 configuration has the associative memory integrated into the 1604-B, the cost of interfacing circuits and buffer registers which are included in the GAP is not present. On the other hand, slightly more extensive modifications to the 1604-B cont circuitry are required than in the case of the GAP. Overall, these two effects balance so that the cost of the A3 configuration is the same as the cost of the GAP configuration or, $336,500 for production copies.

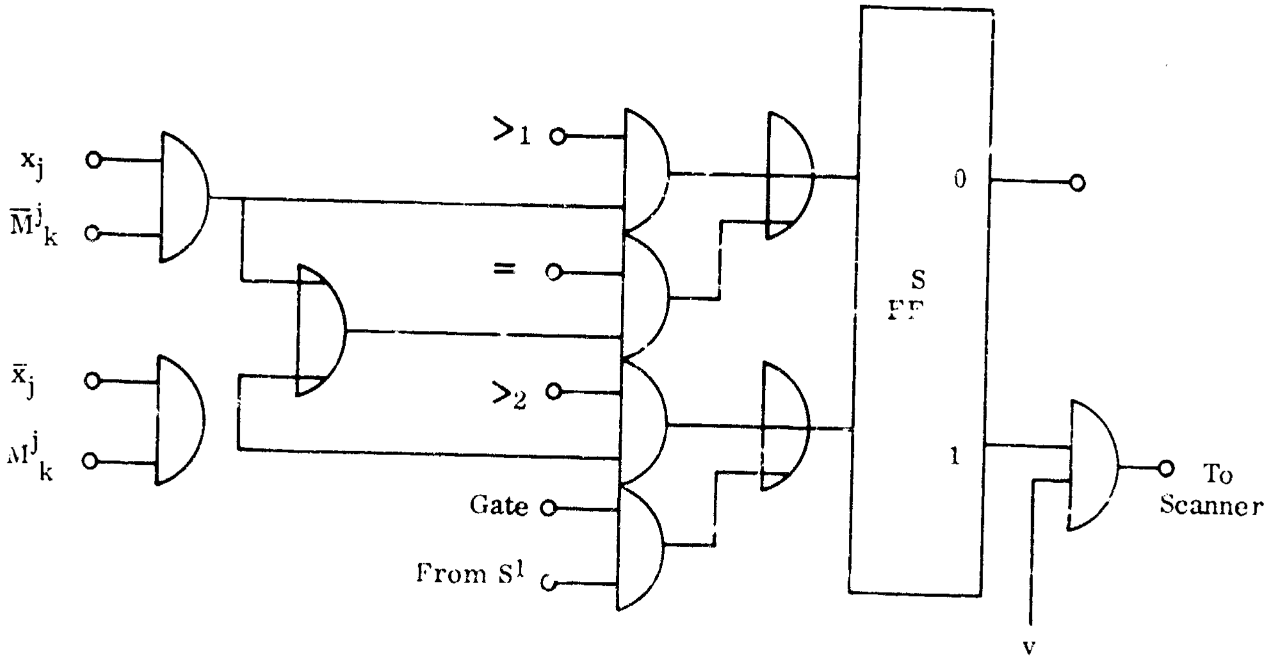


Figure 2-28. A2 Response Store

Multiplying the cost per stage by the number of stages gives a total response store cost of about $45,107.

The cost of the remaining circuits is estimated to be about $15,000. These figures and their total are shown in Table 2-19.

TABLE 2-19. SUMMARY OF A1 COSTS

A1 Component	Cost
Memory Array	$144K
Response Store	45K
Registers, Gates, and Control	15K
TOTAL	$204K

2.7.6 Cost Comparisons

The cost of each of the four configurations is shown in Table 2-20. Note that the total A1 and A2 costs are about 61 percent and 64 percent of the Goodyear and A3 models, respectively.

TABLE 2-20. COST COMPARISON OF AMs

	Goodyear	A1	A2	A3
		(Costs shown in thousands)		
Memory Array and Associated Electronics	$144	$144	$144	$144
Response Store	152	45	50	152
Register, Gates, and Control	40	15	20	40
Total	$336	$204	$214	$336

Note: A3 Does Not Use:

- Instruction Register
- Buffer Register I/O
- Address Register

A3 Requires:

- Decoding Matrix
- Translator
- Sequence Control

2.8 COMPARISON OF CONFIGURATIONS

This report has developed and studied four associative memory, 1604-B hybrid configurations. The four configurations represent three basically dissimilar approaches to the design of hybrid associative memory configurations. The three approaches are represented by the A1 and A2 configurations (which are basically similar), the GAP configuration, and the A3 configuration. It is difficult, if not impossible, to illustrate the differences in approach, implementation, and capability resulting via a simple side-by-side comparison chart. Rather, a full understanding of the design, intention, and functioning of these configurations is necessary to appreciate their different capabilities. For example, consider the A3 configuration. As described above, the A3 configuration and the 1604-B do not provide a core-to-core block transfer or move instruction. Yet,this study has shown that because of the small size of the associative memory, such an instruction would be invaluable in the processing involved in sea surveillance. Such a relatively small difference in configurations, i.e., the presence or absence of a move instruction, cannot be demonstrated adequately by comparison tables, but can make a world of difference in the overall efficacy of a given application. See Table 2-21 for transfer rate of the 1604-B - A3 move instruction.

Because the A2, A3, and GAP configurations each represent a different organizational approach to the employment of identical technology, and because they have greatly different inherent capabilities and their costs are radically different, each should be evaluated separately against different applications. Only in the light of a given application can one be called better or worse than another. Evaluation against applications also discovers minor shortcomings (such as the move instruction for A3) whose removal would greatly enhance a given configuration in a given application.

Some gross physical and organizational differences in the four configurations are displayed in Table 2-22. The following paragraphs elaborate on the entries in Table 2-22.

(1) Number of general-purpose response vector registers.
The number of 1,024-bit registers which can receive the results of a search and in which processing of response vectors can be performed. There are three such registers in A3 while the other configurations have one each.

TABLE 2-21. TRANSFER RATE FOR 1604-B-A3 MOVE INSTRUCTION

Starting Address of Source	Starting Address of Sin	Microseconds/word
1604-B Even Core	1604-B Even Core	12.8
" " "	" " "	6.4
" " "	AM Core	4.8
1604-B Odd Core	1604-B Even Core	6.4
" " "	" Odd "	12.8
" " "	AM Core	4.8
AM Core	1604-B Even Core	4.8
" "	" Odd "	4.8
" "	AM Core	8.0

(2) Total number of response vector registers. The total amount of storage in units of 1,024 bits provided for response vectors. GAP and A3 provide three such registers, A2 provides one register plus a 1/32 of a register which is used internally to accomplish processing of response vectors, albeit slowly, and provides for the storage of eight response vectors in the tag bits of A2; A1 has a single response vector register.

(3) Flip-flops per memory word in the response store. Serves as a measure of complexity of the response store. A3 and GAP have two and one-half flip-flops per word, A2 a little more than one-half of a flip-flop per word, and A1 one half of a flip-flop per word.

(4) Response vector shift. Indicates whether or not a response vector can be shifted to allow the processing of fields and adjacent words in a logical manner. A1 does not shift, A2 shifts up or down but slowly, A3 and GAP both shift down at high speed.

(5) Logic on response vectors. Indicates which of the 16 possible functions of two binary variables can be applied on a bit-by-bit basis between a present search and previous results or in some cases between stored response vectors. A1 provides no logic per se but does provide for "anding" an equal search with previous results; A2 provides for all logic functions to be processed in 32 steps, however, slowly; A3 provides a general AND/OR capability among the three response vectors plus additional setting and resetting capabilities; GAP provides a restricted AND/OR function between results of a current search and previous results.

(6) Use of response store. Indicates what use the programmer may make of the results of a search or sequence of search stores in a response vector. In A1 the response store only conditions 15-bit addresses for retrieval from A1; in A2 the response store conditions the retrieval of addresses or words stored in the associative memory; in A3 the response store serves to define a 15-bit base execution address which may be used in any manner available to the 1604-B (including the immediate retrieval of addresses or data, indexing, or indirect addressing); in GAP the response store marks words for retrieval or (because no prefix register is employed) positions numbers in associative memory for retrieval.

(7) Step-by-step processing of responders. Indicates whether or not the 1604-B can process responders individually and in order from the associative memory or whether it must retrieve a block of responders and perform its own program loops. A1, A2, and A3 allow step-by-step processing, while the GAP does not.

(8) Number of hardware searches. Indicates the number of different searches for which the hardware and sequencing are provided in associative memory. A1 and A2 implement four different searches; A3 implements the ten basic searches of the GAP plus four slow variations of max, min, next higher than, and next lower than: the GAP implements the 10 basic searches plus the between limits search.

(9) Number of search instructions. Indicates the number of instructions which directly cause or control searching of associative memory. Each configuration provides one basic instruction for each search plus modifier fields. The modifier field control is most extensive and powerful in A3.

(10) Number of direct data manipulating instructions. A somewhat subjective count of the number of instructions available in the configuration (excluding 1604-B instructions) which directly manipulate useful results as opposed to loading or unloading of blocks of data or results. A1 provides four such instructions; A2, 13 such instructions; A3 provides 56 such instructions plus numerous modifier fields (many of these instructions are the result of incorporating the full set of operand type 1604-B instructions in associative mode); GAP provides five such instructions plus modifier fields.

(11) Number of housekeeping and load/unload instructions. The number of instructions provided to manipulate the special requirements of the associative memory or to load or unload blocks of data to it. A1 provides 22 such instructions; A2, 34; A3, 24 plus modifier fields; and GAP, 27 plus modifier fields.

(12) Number of tag bits. The tag bits are provided in addition to the 48 data bits of 1604-B word size, and conditions may be specified upon them for searching. A1 and A3 generalize the busy bit of the GAP to produce true tag bits. The A2 machine has eight tag bits with some powerful processing provided for them.

(13) Tag control on load/search. Indicates whether or not the configuration provides for a selective setting of the tag bits when loading and for selective searching of them. The A1, A2, and A3 configurations provide full tag control while the GAP configuration employs a busy bit which must be present for a search to be satisfied.

(14) Source/sink for load/unload. Indicates the place from or to which data may be loaded or unloaded to the associative memory. All configurations, except A3, load and unload only to 1604-B core. The A3 configuration, because the associative memory becomes part of the 1604-B core memory, may load or unload to any peripheral device or transfer data to other sections of 1604-B core.

(15) I/O load (1604-B cycles)/word transferred (in blocks or singly). The load imposed on the 1604-B for each word transferred to or from associative memory in block transfer operations or in single word transfers.

(16) AM instructions per 1604-B word. The number of commands for associative memory which can be contained in a 48-bit 1604-B word. The A1 and A2 configurations each are controlled by EXF codes which are stored two to a word or by INT or OUT instructions which are also stored two per word. The A3 configuration uses a very powerful command code which packs two commands per word. The GAP configuration uses a one command per word format.

(17) Comparand and data shifting. Can the comparand be circular shifted before a comparison and its masking, and can data transferred to or from associative memory be shifted in a like manner? A1 and A2 provide no shifting; A3 and GAP provide shifting.

(18) Mask control on transfer/search. May transfers and searches be masked under programmer option or is the mask always active? In A1, A2, and GAP, the mask is always active, while in A3 a control bit in each pertinent instruction provides for suppression of masking.

(19) Shift control on transfer/search. Can the shifting of comparands and data be controlled or is it always present as mask control? As in mask control, A1, A2, and GAP provide no control; A3 however, provides this feature.

(20) Resolver resettable. Is the response resolver resettable during a sequence of responder processing so that different vectors may be processed as needed? Only A3 provides this feature and it also provides for automatic handling of resolver settings when processing vectors for I/O operations.

(21) Configuration. A1, A2, and GAP are peripheral devices, with GAP featuring independent programming sequences via a special direct memory access channel. The A3 is a fully integrated configuration.

(22) Cost relative to GAP. See Table 2-22 for the configuration comparisons.

2.9 COMMENTS ON THE 1604-B

2.9.1 Introduction

This paragraph serves to document some reservations AUERBACH Corporation has about the validity of data processing applications on the 1604-B computer. Our basic reservation is very simple: the 1604-B, although a fine computer of excellent design, has not been designed for data processing applications and is not representative of the best available in today's market. The Control Data 1604 is basically a scientific computer; it is in fact almost the last of the large-scale machines designed specifically for business or scientific application. It is necessary, therefore, to enter a caveat. This report attempts to evaluate the very latest available in technology, an associative memory, in a hybrid configuration with a somewhat outdated and ill-suited computer. The results obtained must be reviewed in this light, for clearly the use of a more modern data processing-oriented machine would have a significant effect on the goodness of a solution for the sea surveillance problem.

The following paragraphs discuss some points which lead to AUERBACH's basic reservations on the 1604-B.

2.9.2 Input-Output

The 1604-B input-output system is, by today's standard, both inefficient and cumbersome, specifically:

(1) The 1604-B requires two memory cycles for each word transferred in or out via the I/O system. True, this is

better than the three cycles required by the 1604, but not nearly so good as the one cycle required by the 1604-A or machines like the IBM 7090.

(2) The input-output organization of the 1604-B is designed to handle I/O blocks of perspective size. Reading and writing variable length records is difficult and cumbersome.

(3) No scatter-gather facility and therefore no data skipping or suppression facility is provided in the 1604-B I/O system. Scatter-gather I/O systems are invaluable when processing large data bases of variable length files.

2.9.3 Interrupt

The 1604-B has at best a rudimentary interrupt system. By modern standards it is totally inadequate. It provides no separate masking or priority facilities for interrupts from peripheral devices; such facilities are invaluable when trying to maintain maximum access rates to devices such as files. The 1604 interrupt system requires a cumbersome program to determine what has caused an interrupt (and in some cases it is impossible to determine what has caused an interrupt). In other cases, at least in the 1604, it is possible to miss interrupting conditions (i.e., not have the interrupt hardware recognize that the condition has occurred). Although this has been corrected in the 1604-A, its status in the 1604-B is unknown.

2.9.4 Instructions

The instruction set of the 1604-B is not specifically suited to data processing and data manipulation operations. Specifically, there are bit handling instructions, no character handling instructions, no block transfer instructions, no variable length character string instructions, no translate instruction, and no facility for decimal-to-binary conversion and vice versa.

2.9.5 Peripheral Equipment

The disc file employed in the configurations studied here was chosen by RADC to be the CDC 818 disc file system. This system is an older design and cannot compete favorably with such systems as the IBM 1302 which is available today. Since a large data manipulation file processing problem is highly dependent upon the space available and the speedy access to data contained in mass storage, the use of the CDC disc file surely biases the results of this study.

TABLE 2-22. A COMPARISON OF CONFIGURATIONS

ITEM	A1	A2	A3	GAP
1. Number of General Purpose Response Vector* Registers	1	1	3	1
2. Total Number of Response Vector Registers	1	1 plus 1/32 used internally plus 1604-B Core plus 8 in tag bits. (slow)	3	3
3. Flip-Flops per Memory Word in Response Store	1/2	17/32	5/2	5/2
4. Response Vector Shift?	No	Yes (+ or -) relatively slow	yes (+)	yes (+)
5. Logic on Response Vectors?	No (AND Equal search)	All (slow)	AND, OR (general)	AND, OR (restricted)
6. Use of Response Store?	Mark addresses for retrieval	Mark addresses or words for retrieval	Define a general base execution address	Mark words or position numbers for retrieval
7. Step-by-Step Processing of Responders?	Yes	Yes	Yes	No (must block transfer to 1604-B)
8. Number of Hardware Searches	4	4	14	11
9. Number of Search Instructions	4 plus modifier fields	4 plus modifier for logic	14 plus modifier fields	11 plus modifier fields
10. Number of Direct Data Manipulating Instructions	4	13	56 plus modifier fields	5 plus modifier fields

* Amdahl, G.M., et. al. Architecture of the IBM System/360. IBM Journal of Research and Development, Volume 8, no. 2, April, 1964.

TABLE 2-22. A COMPARISON OF CONFIGURATIONS (Cont)

ITEM	A1	A2	A3	GAP
11. Number of Housekeeping & Load/Unload Instructions	22	34	24 plus modifier fields	27 plus modifier fields
12. Number of Tag Bits	1	8	1	0 (has busy bit)
13. Tag Control on Loading/Search	yes	yes	yes	no
14. Source/Sink for Load/Unload	1604-B core	1604-B core	Any peripheral device or 1604-B normal core	1604-B core
15. I/O Load (1604-B Cycles) / Word Trans. (in blocks or singly)	1/1-1/2	1/1-1/2	2/NA	1/2 or 3
16. AM Instructions per 1604-B Word	2 (EXFs)	2 (EXFs)	2	1
17. Comparand and Data Shifting?	no	no	yes	yes
18. Mask Control on Transfer/Search	no	no	yes	no
19. Shift Control on Transfer/Search	no	no	yes	no
20. Resolver Resettable	no	no	Yes, also automatically for I/O transfers	no
21. Configuration	Peripheral device on Channel 7	Peripheral device on Channel 7	Fully integrated in 1604-B structure	Peripheral device with independent pgm. sequencing on special DMA Channel.

TABLE 2-22. A COMPARISON OF CONFIGURATIONS (Cont)

ITEM	A1	A2	A3	GAP
22. Cost Relative to GAP (including 1604-B modifications but not 1604-B or peripheral equipment)	59%	62%	95-105%	100%
23. Special Comment	A1 is a stripped-down machine for study purposes and training only	A2 is about the minimum workable configuration which is balanced for the AM technology employed	A3 retains all normal 1604-B functioning and adds associative features. A MOVE (block transfer) instruction in the 1604-B would be very valuable	GAP is an ambiguous configuration - not yet a multi-processor, a little more than a peripheral device. Much better coordination and synchronization facilities are needed. The design should be more efficient using 1604-B memory and should allow more detail control by the 1604-B.

SECTION III. THE SEA SURVEILLANCE PROBLEM

3.1 INTRODUCTION

The purpose of a sea surveillance system is to keep a close watch or maintain a close supervision (a dynamic surveillance) of all ships within a geographical area. The objectives of such a system include the following:

- Cognizance of all ships entering the area.
- Identification of unknown ships within the area.

It is desired to achieve these objectives to:

- Strengthen national security.
- Enable rescue operations and issue warnings in times of national emergency and acts of God.

A sea surveillance system is in part, a system under the cognizance of national security agencies and is therefore classified both on a need to know basis and on a security classification. For this reason, a comprehensive problem statement was not obtainable from outside sources. Consequently, the sea surveillance problem given in this section is perhaps different from the actual working problem.

The very nature of the objectives and the motivation for these objectives admits many variable situations in a real life sea surveillance system. Part of the problem then is to define the problem. It was not the attempt of this study to "solve" the sea surveillance problem and design an associative memory system for the determined solution. Rather, it was desired to formulate "a solution" for "one" sea surveillance problem and to assess the associative memory as an aid in effecting this solution.

The prime source of information used in defining this "one" sea surveillance problem is the document "Sea Surveillance Data Base Representation as Test Vehicle".* This document does not define a problem; however, it describes an unclassified sea surveillance data base, presents input messages, and lists a set of questions concerning the stored data. One way the problem may be defined is to view of some of the processing

* Sea Surveillance Data Base Representation as Test Vehicle - Prepared by IBM for the Office of Naval Research, Washington, D.C., Contract NONR 4420(00). June 30,1964.

functions. These are:

(1) Admit and store in the data base a message about an item.

(2) Maintain the data base as a result of the input message.

(3) Retrieve that information to s. tisfy the questions given in the source document.

Thus, for all practicable purposes. the problem defined in a macroscopic sense, is a data storage and retrieval problem.

As in several other intelligence studies, there exist an abundance of available functional tasks that may be effected depending upon the scope of the problem defined. It is probably easier. as an initial cut, to enumerate certain problem areas that may be solved by other sea surveillance systems, but will not be considered in this study. These are:

(1) Multi-leveled information classed on:

   (a) Security

   (b) Need to know

   (c) Quality

   (d) Veracity

   (e) Priority

(2) Formation of intelligence information by deduction and induction by probabilistic and/or graph theoretic methods.

(3) Historical data.

Thus, the system considered in this report will concern itself with input information of a uniform quality with no security, priority, or need to know classifications. The system will not maintain obsolete data and will not be concerned with the formation of intelligence data.

3.2 DATA BASE CONTENTS

The data base contains descriptive information about items of interest. An item is defined to be a ship, a personality. or a port. The descriptive information is termed descriptors. A descriptor may be thought of as a value of one of the items' attributes. For example, a ship has such attributes as speed of advance, crew complement, weight, etc. For each attribute, an input message may describe the ship by specifying some of these attributes. Additionally, an item may be a descriptor for another item. For example, a personality aboard a ship may be considered as one of the ship's descriptors. A full description of all items and attributes and their relationships is contained in Paragraph 4.3.5. The information stored in the data is current.

3.3 INPUT MESSAGE PROCESSING

One method of defining a problem is to give its scope. This approach will be used in this and subsequent paragraphs. Every input message is construed to be a message about an item or items which is described by a set of descriptors. If the message concerns more than one item. it is assumed that this message may be the genesis for a set of messages, each about one and only one item. Certain assumptions are made regarding certain items and descriptors. These are:

(1) Every message about a ship contains the ship's latitude-longitude positional descriptors. This holds even if the "name" of the ship is not known.

(2) Every message about a personality is related to a ship or a port.

In addition to these assumptions, certain assumptions affecting the system design must be stated. Theso are:

(1) The data base, that is, the set of items and attributes, is not constant. Therefore, it must be possible to:

(a) Add and/or delete items.

(b) Add and/or delet attributes.

(2) The data base is of sufficient bulk that it must be stored on an auxiliary storage device. The data stored will be required randomly; therefore, a disc file will be assumed rather than a tape file. This disc file assumed is the CDC 818 and the associated CDC controller.

An input message must be incorporated into the data base. Several situations arise. The following ad hoc rules are followed:

(1) If the message concerns a known ship (the descriptor for its name is given) and

(a) a record does not exist about this ship in the data base: then a record is created from the message and entered into the data base. The ship is construed to be entering the area of surveillance.

(b) a record exists (i.e., the ship already exists in the area of surveillance) then the following may occur:

1) A descriptor is given for a new attribute previously not valued. This descriptor is entered into the existing record as an added piece of information.

2) A descriptor is given which confirms (is identical to) an existing descriptor.

3) A descriptor is given for an attribute which differs from a previously given descriptor for the same attribute. In this case, the "class" of the attribute is examined. If the attribute is one which can not change; e.g., the ship's propulsion or cargo or armament characteristics, an error is noted. However, if the attribute is one which may admit descriptor changes, position, speed of advance, etc., the change is inserted.

(2) If the message concerns an unknown ship (the descriptor for its name is not given) the system will advise the user who may use the query subsystem to aid in determining the merging "unknown" ship records. Otherwise, the message is treated as a "new" ship entering the system.

(3) If the message concerns a personality, a ship must be associated with the personality in the record. If the personality is not currently represented in the data base, a record is created for him. This record is "linked" to the ship's record (and conversely) if the ship exists. If the ship does not exist, a new ship's record is created.

(4) If the message concerns a port, it is processed similarly to the processing used for a message about a ship.

3.4 MAINTENANCE OPERATIONS

The complexity of maintenance operations is dependent upon the physical organization of the data base. Indeed, these operations usually dictate the physical organization. (It may be considered that the logical data base organization is biased toward the retrieval operations). However, the scope (not the complexity) of the maintenance operations is determined by the rules governing the disposition of input messages. In addition, certain maintenance operations result frɔm assumptions regarding the addition and deletion of data. Then from the preceding paragraphs, the following maintenance operations are to be effected:

(1) Create a new record which may require more than one disc block.

(2) Insert a new attribute-descriptor pair into an existing record.

(3) Delete a complete record.

(4) Delete attribute-descriptors from an existing record.

(5) Maintain correct cross-references when required between data records.

(6) Maintain file (a collection of homogeneous records) integrity.

(7) Maintain directories and dictionaries when data is added and/or deleted.

3.5 QUERY OPERATIONS

One of the major objectives of a sea surveillance system is to retrieve and present stored information and/or values of a function with stored information arguments. The commands to retrieve and present this information are termed queries. The queries considered explicitly in this study are:

(1) How many ships are within "r" miles of point "p" ?

(2) Are any U.S. submarines in area ______________ ?

(3) How soon can DD-789 reach Bermuda under normal speed of advance ?

(4) Where is Admiral ________________ now ?

(5) Are any ships scheduled to be in the projected vicinity of Typhoon Dottie the next few days ?

(6) Nearest (in time) ship/aircraft with doctor aboard to point x, y?

(7) What ships with long-range radar could be in area ____ _______ by 2400 tomorrow ?

(8) Ship sinking at ____________. Which ships can be there first?

(9) Nearest ocean-going tug to sinking ship ?

Note that these queries do not require all 108 attributes found in the data base. This does not mean that attributes not required should be eliminated; rather, it implies that the queries are typical and representative of a complete set which may require the entire data base. In like manner, the data base given may be considered as representing the required data base for some time interval during its development and use. Thus, neither the data base nor the set of queries should be considered in final form.

The queries given have descriptor values inserted for attributes. Perhaps it would be more meaningful to have listed these queries on an item-attribute level. For example:

- What ships (by type and/or country of registration and/or weight and/or ...) are in an area x ?
- How soon in time can a ship ________ reach the port _________ (or point x, y) under normal (or maximum) speed of advance ?
- Where is personality ___________ now ?

There are several methods to admit query processing. At one extreme the processing required for each query may be completely encoded in machine code, stored on the system's tape, and retrieved by a call. This call (perhaps the query's name or number) may be considered as a query language. Since neither the data base nor the queries may be considered complete at any one time, it is impossible, or at least time-cost prohibitive to follow this possible method of effecting queries. Examination of the given queries to determine some general features shows that each query requests statistics about or descriptors of a set of attributes of one or more items. The set of attributes and/or items are assumed to satisfy some conditional statement. If is felt that the technique needed to directly supply answers to each query is not explicitly known at this time. The user of the system must be relied upon to state the query in a permissable way to retrieve the pertinent data and then interpret the results.

A user's query language is one which attempts to provide a strong linguistic capability together with convenience of use. Such languages are either translated into machine code or command list for processing by a compiler-type pre-processor. Such languages may be regarded as parameters controlling processing as by an interpreter.

In this study we have elected to translate (Paragraph 7.2), a user's query language into an internal language in Polish prefix form. This internal form is then "compiled" as object code by a pre-processor (Paragraph 7.3), into a command list form. A Run Routine (Paragraph 7.4) then "performs" each element of this command list in an interpretive fashion. The Run Routine is controlled by an executive level routine which interleaves input-output (Paragraph 7.6) requirements with portions of the Run Routine. The executive level system is called the Controller (Paragraph 7.5).

It is beyond the scope of this study to formulate a user's query language which is amenable to associative memory processing. Also, it is deemed inappropriate to consider translation of an existing user's query language by use of an associative memory since the existing language was formulated to be translated by non-associative memory processing. The only restriction placed on the user's language is that it can be processed into Polish prefix form. The language shall not be specified further; however, not all current query languages meet this restriction.

The functions of the Query Language Translator include:

(1) Resolve all synonyms into one term.

(2) Eliminate redundant and "noise" words.

(3) Resolve functions and operations into Polish prefix canonical form.

The pre-processor performs the following:

(1) Converts the Polish prefix form into a command list form.

(2) Equates operations' outputs to subsequent operations as input variables.

(3) Determines operations' ranges to order operations in an optimum processing order.

(4) Determines data used by more than one operation to avoid duplicate input operations.

(5) Assigns unique tags to all inputs and outputs to minimize data transfers.

The function of the Controller is to:

(1) Interleave processing control between the Run Routine and the Input Routine.

(2) Assign and allocate 1604-B core memory to variables in the command list.

The purpose of the Run Routine is to:

(1) Perform certain operations set by the Controller in an interpretive fashion.

The function of the Input Routine is to:

(1) Reorder a list of disc addresses into an order which results in the minimal disc access times.

3.6 SYSTEM CONSTRAINTS

Several assumptions regarding data sizes have been made. These ar

(1) There is no requirement for a study involving dynamic memory allocation.

(2) There will always exist sufficient 1604-B core memory to:

(a) Read a complete input message.

(b) Read a complete data or directory record and form a new record.

(c) Hold both variables in entirety for one query operation.

The last assumption does not impose a restriction on data size regarding the associative memory. The system allows for the fact that there may be more data than can be stored and processed in the associative memory at one time. A routine, called the AM Dispatcher, functions to partition associative memory input lists into proper size segments. This dispatcher then controls the data flow between the two memories and the processing flow between the input-output operations and the functional routine. The AM Dispatcher assumes that such ordered segment processing yields the same results as if the segmenting did not occur. The definition of "ordered processing" depends upon the functional operation. Suppose that the two lists being operated upon were L_1 and L_2 partitioned into L_{ij} $(1 \leq j \leq J)$ and L_{2i} $(1 \leq i \leq I)$. Then ordering may be such that i and j are varied simultaneously or that one subscript is varied for a fixed value of the other. The first ordering is used if it is desired to form a list showing distances between points given in L_1 and L_2. The second ordering is used if it desired to find the intersection of L_1 and L_2.

3.7 STATISTICS

Throughout this study, it was necessary at certain times to make estimates of record sizes, number of computer words per descriptor, number of responders per associative memory load, etc. These estimates or statistics are then used to determine such things as processing times, number of responders, size of data (number of words) transferred, etc., which are used in turn to evaluate the associative memory configuration and/ or differences in system design. These statistics are not too meaningful when divorced from the content in which they are given. Therefore, such statistics are deferred in this section.

3.8 EVALUATION

Throughout this study, criteria were used to evaluate different associative memories. These evaluators may be dichotomized as quantitative or qualitative measures. For example, a quantitative measure is the time required for each associative memory instruction,and a qualitative measure indicates the difficulty in flow-charting a problem whose solution uses an associative memory. The quantitative evaluators may be considered to be objective measures, whereas the qualitative evaluators are less objective. Evaluators used at any one time are dependent upon the particular application to which they are applied. For example, some evaluators used in a hardware oriented analysis would not be applicable in a software analysis.

3.8.1 Quantitative Evaluators

This class of evaluators may be divided into three classes depending on whether the quantity measured is time, costs, or neither time nor costs. It is also possible to combine two or more of these evaluators to see the overall result. For example,time may be considered as an independent variable with costs as a dependent variable.

3.8.1.1 Time Evaluators

(1) Time for each associative memory instruction.

(2) Time for subroutines.

(3) Time required for 1604-B to set up associative memory processing (data transfers, etc.).

3.8.1.2 Cost Evaluators. A complete method of developing and evaluating hardware costs is presented in Section II.

3.8.1.3 Other Quantities (Neither Time nor Costs).

(1) Number of associative memory and 1604-B instructions used (and not used) and frequency of each instruction.

(2) Amount of data that is read into and out of the associative memory and frequency of transfers (data transmitted between 1604-B and associative memory).

(3) Number of associative memory searches required to complete each associative memory instruction and macro instruction.

(4) Ratio of total data read into the associative memory and the number of responders.

These quantities are dependent upon the particular application. Indeed, for certain phases of an application, these quantities may differ appreciably.

3.8.2 Qualitative Evaluations

The qualitative evaluators are more properly qualitative evaluations of the hybrid configuration with respect to certain topics. Each of the evaluations to be presented is worthy of a more thorough investigation than it will be possible to achieve in this study. In other words, each evaluation will be made in view of the results obtained in this investigation rather than attempting to complete a definitive evaluation. Rather than create a specific section, evaluations are interspersed throughout the report.

3.8.2.1 Configuration. A hybrid configuration consists of a particular associative memory, a general-purpose computer, and mass storage devices - both sequential and parallel. This evaluation would attempt to show where general-purpose computer features not found in the 1604-B and mass storage device features not found in the CDC 1619 control unit and 818 disc file, result in more or less "better" systems. In short, the objective of this evaluation is to determine the best general-purpose computer and mass storage device features, where "best" is defined with respect to the particular associative memory.

3.8.2.2 Data Organization and Formats. This evaluation will attempt to indicate the impact of an associative memory upon data organization and data formats. In a general-purpose system, data organization is dependent on (among other things) how the required functional operations are effected and on the data storage system. Since the method of achieving a functional operation may change in a hybrid associative memory system, it follows th the data organization and format previously assumed may also change.

3.8.2.3 Problem Analysis Techniques. Current techniques in this area include the actual encoding languages and such devices as flow charts and algorithms. This particu evaluation will consider the effect of an associative memory on flow charts and algorith For example, current flow charts indicate decision boxes, operation boxes, and informat and control flow. It seems feasible, at this time, that the associative memory may pen the marriage of certain operation and decision boxes, as well as eliminate the need for flow charting some functions (such as list searching) to the level formerly required.

SECTION IV. DATA BAS[illegible]

4.1 INTRODUCTION

The purpose of this section is to provide a detailed description of the data base and file organization utilized as a model in the system. The data base was obtained from the Office of Naval Research (ONR) document.* The logical structure of the file organization and the physical structure of the data base mapped onto the disc were designed by AUERBACH. The choice of the machine organization consisting of discs, associative memory, and general-purpose computer was selected by AUERBACH. Specific hardware components (e.g., the CDC 818 and 1604 computer) were specified by RADC.

The rationale behind the specific choice of the file structure is first developed in Paragraph 4.2. The specific data contained within the system is developed in Paragraphs 4.3 and 4.4, which present the detailed statistics associated with the sea surveillance problem. Paragraph 4.4 presents conclusions relevant to the role that an associative memory plays in a computer system for a comprehensive data storage and retrieval system such as the sea surveillance problem.

4.2 FILE STRUCTURES AND FILE DESIGN CONSIDERATIONS

4.2.1 Criteria for File Organization

The data base of the sea surveillance problem consists of data concerning ships, parts, and equipments used in the U. S. Navy. This information is to be accessible to users through direct hard-copy reports requested by means of a query language. Files are designed so that the report information can be used in a systematic, efficient way. Since a specific kind of data is put to more than one use (and sometimes the potential use is partially unpredictable), the file design is always a compromise between those designs that are optimum for competing uses. This compromise is, of course,

* Sea Surveillance Data Base Representation as Test Vehicle prepared by IBM for the Office of Naval Research, Washington, D. C., Contract NONR 4420(00), June 30, 1964.

weighted by the importance of responsiveness requirements of the using programs. For example, there are the (sometimes complementary) requirements for data maintenance (or file updating) and for data retrieval in response to a random query. It is not unusual that a file structured to favor responding to one type of query is poorly suited to respond to another type of query or is inefficient to update.

An optimal file design is one for which the necessary processing can be performed in the shortest time. The processing of large files (too large to exist in core or in a small as-ociative memory) is generally access limited; that is, data movement is the dominating factor in p-ocessing time. Stating it another way, an optimal design groups data according to processing needs so that each access to secondary storage yields a large amount of data actually needed for processing, thereby minimizing the number of accesses.*

Notwithstanding the random access capability of disc storage, data is most efficiently handled in relatively large sequential segments. This is, it is economic to dedicate a large buffer area in core and, once a disc arm is positioned, to transfer as large a segment from that position as can be accommodated.

As far as the above discussion has gone, this is not unlike conventional tape usage. There can be a great deal of similarity in the use of the two media. The above principle is applied in a very straightforward man er in the sequential search strategy necessarily employed in a tape-based system. However, for those tasks that will ultimately require only a small part of the file, the addressing capabilities of a disc system (through the use of a more complex access strategy) make it possible to reduce processing time considerably. This is a regenerative factor since - as a general principle, where fast response is available — users (and processors) will take advantage of it and the system will process smaller chunks of information on a basis that is closer to real time (non-batched). There is another compromise (tradeoff) that is possible here; as already pointed out, the total processing efficiency (throughput) is highest when data is transferred in large segments with a high yield of needed data in each access (hence, a batched service operation).

*Minker, J. 1961. Implementation of Large Information Retrieval Problems. National Science Foundation.

To take advantage of this trade-off possibility (i.e., to have the system function both ways, to furnish quick response with random-access files and to provide high throughput of batched servicing when the service load demands it and the deadlines permit it) is the real challenge of the design of files and their processors.

4.2.2 Strategy of File Organization

Of the two basic types of file processing, updating and query reply, updating usually presents the more consistent and predictable requirements for file organization, often because the input data (transaction records) are obtained from known sources and contain standard data whose format can be controlled. Queries, on the other hand, are by nature unpredictable and make varying demands in terms of search criteria, data to be tabulated, sorting keys, etc. A basic method of utilizing addressable storage for files subjected to these varied processing demands is to organize the file on a primary basis for efficient updating (a relatively difficult task) and to provide a mechanism for linking those records that share an important characteristic from a query standpoint. For non-addressable portions of memory, such as records in a tape file, a standard approach has been to store information in more than one organization, if necessary, so that important types of queries, with short reply deadline requirements, can be accommodated. This may mean, for example, maintaining more than one file with the same basic information, usually sorted according to different keys. Rather than duplicate data, which compounds the updating problem, the use of addressable storage permits non-redundant files with the required organizations provided logically by means of links, lists, or directories, rather than by physical duplication as performed in tape operations.

Due to the physical nature of tape units and tape reels, the updating of tape files is usually done on a batched basis. That is, since the reel is rewritten at the time of update and the amount of time needed to update the tape is almost independent of the number of records modified during the pass, modifications to a file are usually saved (batched) on a transaction tape until it is necessary (or economic) to update the file. The addressing capability of the disc unit, however, makes it possible to update the file as modifications are received without an extreme penalty over batched operation.

A technique called "address randomizing" is sometimes used in random access memories. This technique can usually eliminate the disc access needed for directory retrieval by substituting an address calculation based on a unique record key such as part number. However, since part numbers may be mapped into the same address (or "bucket"), there is a level of storage utilization (usually around 80 percent) at which this technique involves a greater number of disc accesses than a directory approach, due to the fact that the first "bucket" is full and an overflow link to a second is given. (As the number of records approaches the memory capacity, the "chain" of bucket links will increase to large values for some entries.)

In discussing file organization (especially for addressable storage), it is useful to distinguish between the physical and logical organization to define precisely the physical and logical data elements of interest.

A physical element is a hardware recognizable unit of data; for example, a word in core, a tape reel, a segment on tape between tape marks (between "load point" and "end-of-reel" markers on a particular tape unit), a block between gaps on tape or an addressable block on disc, etc. A logical element is a symbolic unit of data such as a string of symbols that is program recognizable due to its position, length, or the control symbols used as punctuation. A logical element may consist of a part of a physical element, several adjacent physical elements, or physical elements that are not adjacent but are address linked in some way. The basic physical (addressable) element on disc will be called a block (other physical units are track and disc). Logical elements will be called fields, records, and files. Just as physical elements are nestled hierarchically (e.g., blocks in a track), so are logical elements as field in a record, and records in a file.

A field is defined as any logical element having a specified format and recognizer. A file is a field that is composed of an arbitrary number of repetitions of a subfield called a record. A record in turn contains fields some of which may be files, etc. A field which is interpreted as an address is called a link. Some addressable records, called items, will refer to external objects that are partially identifiable by formalized (and perhaps coded) index terms called descriptors, which may exist as fields in the record. A descriptor may usually be interpreted as a value of a particular attribute whose range is well-defined. For example, ship name, commanding officer, and port may be attributes of records in a sea surveillance file. The descriptors characterizing a particular record may be shipname: Reuben S. Gomez; commanding officer: Capt. John Jones; and port: Norfolk, Virginia. The term in front of the colon is the attribute and the term after the colon is the attribute value.

4.2.3 Multi-List Organization

The sea surveillance problem must be able to store data concerning many ships and ports and to answer requests for data of various ships in specified areas or with specialized equipment or capability. The variability of the file entry key and the random nature of the service requests make it impractical to use address calculation (or "address randomizing") or sequential processing as the sole means of record retrieval. A basic technique that can be employed is to select several index terms for each record and maintain lists of records that are characterized by each index term. This technique allows many entry points for rapid retrieval compared to the single possibility in address calculations or the slow sequential search approach. (It is possible, and sometimes desirable, to combine more than one approach in a single system.) In a multi-list organization, each index term is a descriptor whose coding may express hierarchic relationships to other descriptors.

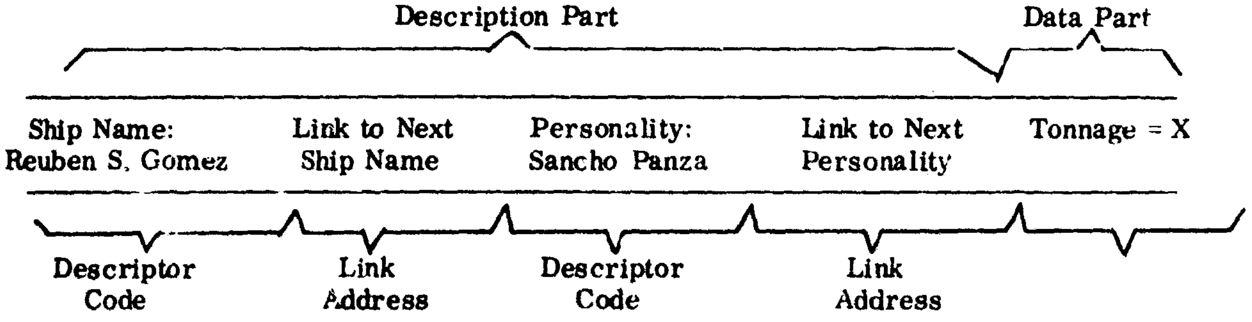
With each different descriptor is associated a list in the memory on which is placed every data item having that descriptor in its description part. Since the item may, and usually does, have more than one descriptor, it may be placed on more than one list. The item will, however, be located in only one physical location in memory. The lists on which a particular item is placed are said to "intersect" at that item. Storage of the item is accomplished by placing it on all lists associated with its descriptors and also entering it into the data file.

It is clear that a list of records (in addressable memory) that have a descriptor in common can be designed in two ways. One way is to have a table of descriptors and, with each descriptor, a table of links to records that have that descriptor. Another way is to have, with each descriptor, a link to the first record with that descriptor, and have that record, in turn, contain a field linking to its successor having that descriptor, etc. These two approaches are discussed in the following paragraph.

In any implementation of the multi-list organization, it is necessary to have a table which translates the descriptor terms (English terms for attribute and value) to the internal code that is used and which may express hierarchic relationships among the descriptors. It is also necessary to implement in some way a search strategy for finding the first record when starting with a given descriptor. One method for accomplishing this is a conventional table lookup. Another has been described as the balanced tree approach.* The design of this system involves a conventional table look-up approach, since the effectiveness of the balanced tree approach depends on a uniform probability of descriptor selection.

4.2.4 Implementation of Multi-List Organization

Records (data items) in a multi-list organized file consist of two parts: a description part and a data part. When implemented, using the record linkage approach, the description part contains the descriptors describing the time, and associated with each descriptor is a link to another record with the same descriptor. The data part of the record contains other information about the item not necessarily linked to other records. Both parts and their components are shown in the illustration.



With this implementation of the multi-list organization on disc, the search of a list of several hundred items is inefficient due to the many accesses required, and may thus negate the advantages of this memory organization scheme. Also, the storage of description and data portions of the item in contiguous locations may become available and must be used. This will result in the need for linking addresses within the item itself which will require an additional memory access which will significantly increase the retrieval time. This latter problem may be relieved by various housekeeping programs incorporated into the storage and retrieval programs.

*Landauer, W.I., 1962. The Tree as a Stratagem for Automatic Information Handling. University of Pennsylvania, Moore School Report 63-15.

Another method of implementing the logical structure of the multi-list organized memory is by sorting the item (consisting of descriptors and date) in one location in memory and the descriptor tables and links in another portion called a "directory." The search for the item is by descriptor, with the search method differing due to the essential characteristics of the lists.

In the directory approach, the item is stored in a data file and the address of the item is stored in the record index for each of its descriptors. The retrieval of an item can be accomplished by finding those addresses that are common to the record indexes of the desired descriptors. A generalized retrieval strategy which permits the determination of the exact records required, with a minimum number of accesses to disc, makes this approach more efficient than the linked multi-list structure approach.

If part of the directory is stored in high-speed memory, the number of accesses to disc for retrieval or storage will be at most the number of descriptors by which the item is stored plus one access for the item. If core space is available, access time may be further reduced by storingoften-used record indexes in core to reduce the number of accesses to disc.

4.2.5 Associative Memory Considerations in File Design

An associative memory (AM) played no role in the previous discussions. The reason for this lies in the fact that the sea surveillance problem is such that

(1) the entire data base significantly exceeds the capacity of the AM, and

(2) the entire data base need not be compared to a set of data which permanently resides in the AM.

Indeed, this was a reason for selecting the sea surveillance problem for the study. Thus, in considering the organization of the data base, a major question had to be answered: How can an AM be utilized if only a small amount of data can be placed in it at any one time, and the main data store is on disc? In this type of a situation, the time to transfer data from disc to core to AM on input, and from AM to core to disc on output, becomes significant particularly when the data being manipulated in the AM does not reside there for any great length of time. A major reason for developing the A3 configuration was to attempt to eliminate the data transfer time between AM and core.

Since the data base must be stored on disc, it is necessary to be able to focus in on the desired d^ta rapidly without requiring a search through the entire data base. This is necessary whether or not an AM exists within the system. Since it is uneconomical to store the entire data base to retrieve or maintain a set of data records, an indexing scheme has been developed and is described below. As noted in Paragraph 4.2.4, the indexing scheme is a multi-list file structure in which the linkage to records is extracted from data records and placed in directories. From a retrieval basis, this organization allows data to be retrieved faster than that of a multi-list file structure with linkages in data records. This retrieval speed pertains whether or not an AM is utilized. Thus, the overall organization of the file structure utilized to access data when the AM is in the system has not changed.

An AM can aid in freezing the format of data contained either within a block of data that can be loaded entirely within the AM, or the format of data in a record. Because of the ability of the AM to search on the name of the item, its specific location within the AM is unimportant provided that the data can be identified. A major reason for a fixed field type of record is that one can go directly to the data required without having to scan the entire record. On the other hand, when significant gaps of information exist within a majority of the data records, this type of an organization indicates that a large amount of excessive storage is wasted.

The realization that intelligence data records could not be equally "rich" in data field values, because of the fragmentary nature of intelligence data, led to two major decisions. In order to conserve disc storage space and therefore decrease disc read time, the records were to be stored in as packed a form as would lead to a reasonable system. Because intelligence records would not always have a data field reported, space could be saved by using storage space only for reported fields. Hence, the attribute-descriptor form of presenting data was selected for representing data fields, and only reported fields were included in the record. One further result of the fragmentary nature of intelligence data and the effort to conserve space was the decision to permit individual fields to be of variable length in the data records themselves. This decision introduces some interesting aspects to the file maintenance subsystem, although it undoubtedly corresponds to the manner in which intelligence data is presented to the system.

Had a fixed field format been selected, extra disc storage would be required. On the othei hand, a fixed field record obviates the need for an associative memory since one can pick up the data directly. Thus, although the AM does not play a major role in the accessing of data from disc, it plays a role in the formatting of data within records and eases the requirement for rigid formats and for sequencing data in records.

4.3 MULTI-LIST FILE STRUCTURE FOR SEA SURVEILLANCE PROBLEM

4.3.1 General Considerations

This paragraph is concerned with the specific file structure that is utilized for the design of file structures described in Paragraph 4.2. The basic problem is to design first a file structure which will take advantage of the random access capability represented by disc storage and, second, to structure the retrieved data so as to manipulate it in an associative memory. If, for every query, it were necessary to retrieve every record from disc and then determine whether or not the record satisfied the query, it would be more efficient to utilize tapes.* On the other hand, if for every query one descriptor (say Federal Stock Number) was specified, it would be more efficient to develop a routine which mapped Federal Stock Numbers into a disc address in which the appropriate record would be stored. It is clear that for sea surveillance information, neither of these two conditions applies.

To effectively utilize disc systems, it is necessary to design a system to assure that whenever information is retrieved from disc, a minimal number of accesses to disc is required. That is, it is desirable to focus-in on the information set that is required rather than to have to scan the file by going into the disc to determine if the appropriate information has been obtained and, if not, to retrieve another disc address. The file structure which is outlined satisfies the focusing-in process. It is a multi-list file structure in which the links have been separated from the data.

4.3.2 Multi-List File Structure

The multi-list file structure consists of the following two major structures:

(1) The directory

(2) The data base

*Minker, J. Implementation of Large Information Retrieval Problems.

The directory serves the purpose of focusing-in on the appropriate records in the data base. The directory is made up of a file location directory (D = Directory) and a file directory. The data base is the collection of all data in the system. The organization of the files is shown in Figure 4-1.

4.3.3 Definitions

Before specifying the details associated with the multi-list file structure, some definitions should be presented. The actual sea surveillance data is organized into a set of groups termed files. A file consists of a set of homogeneous records. By homogeneous is meant that the data within the record pertains to a set of defined terms called descriptors. The subject that the record is describing is called an item. Thus, a record in a file contains a list of descriptors relative to a particular item. For example, the PORT file contains records about ports. The item of a port record is the logical name. The descriptors in the port record include the specific port name code, port name, and port position, to mention several eligible descriptors for the record. A descriptor relates a value to an attribute. Some attributes in the port record are port name code and port name. Associated with each record is a logical name. The logical name is a unique tag which identifies the record. Every record contains a logical name. The logical name of the record is identical to the item.

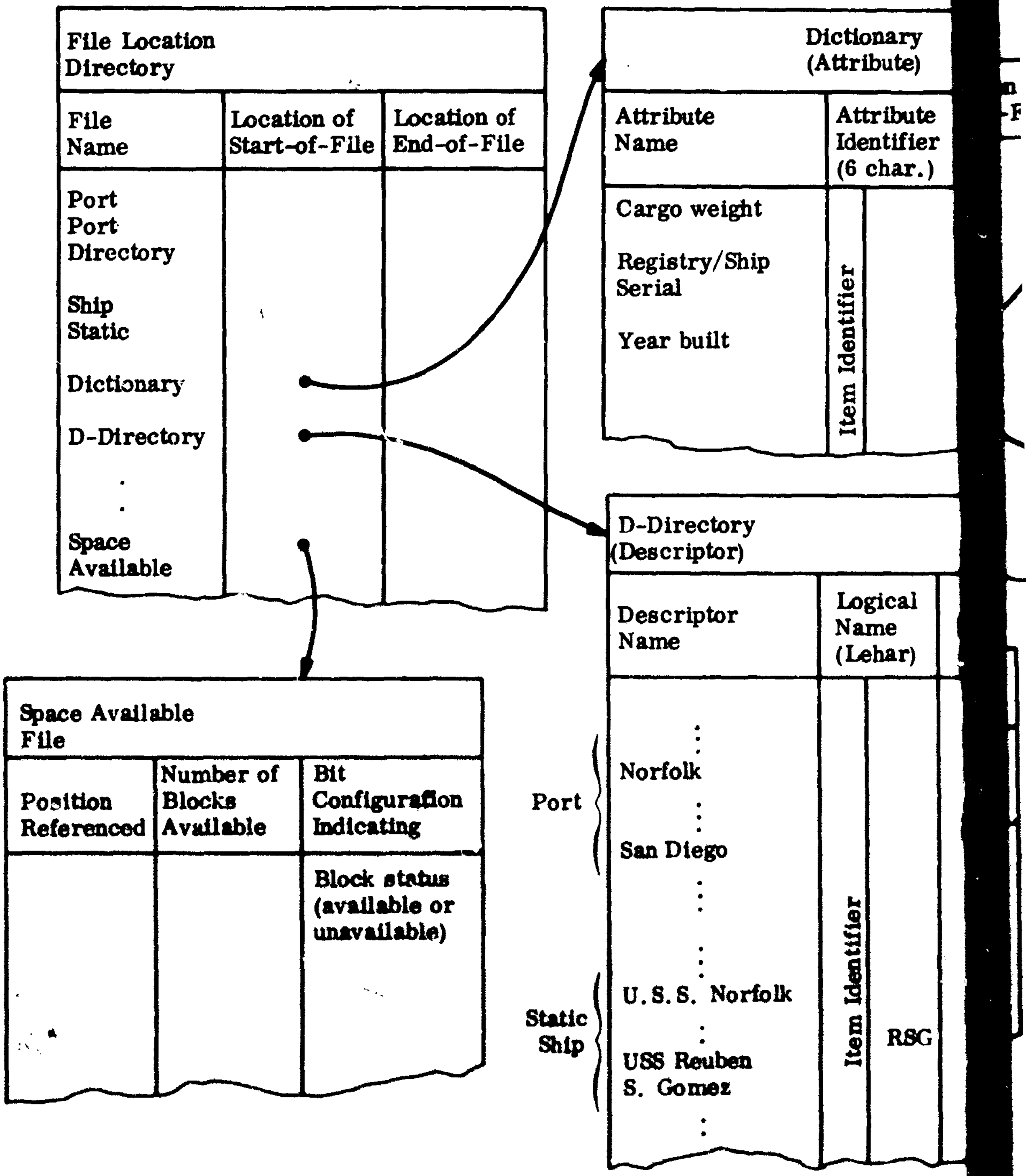
4.3.4 Data Files

The following constitute the totality of data files within the sea surveillance system covered in this study.

(1) Port
(2) Personality
(3) Ship (Static)
(4) Cargo Characteristics
(5) Operating Characteristics
(6) Propulsion Characteristics
(7) Weapons Characteristics
(8) Ship (Dynamic)

4.3.5 Data File Records and Attributes

A record in each of these files consists of a list of descriptors (i.e., values for certain attributes). For each file, the following attributes may be



File Location Directory

File Name	Location of Start-of-File	Location of End-of-File
Port		
Port Directory		
Ship Static		
Dictionary		
D-Directory		
. .		
Space Available		

Dictionary (Attribute)

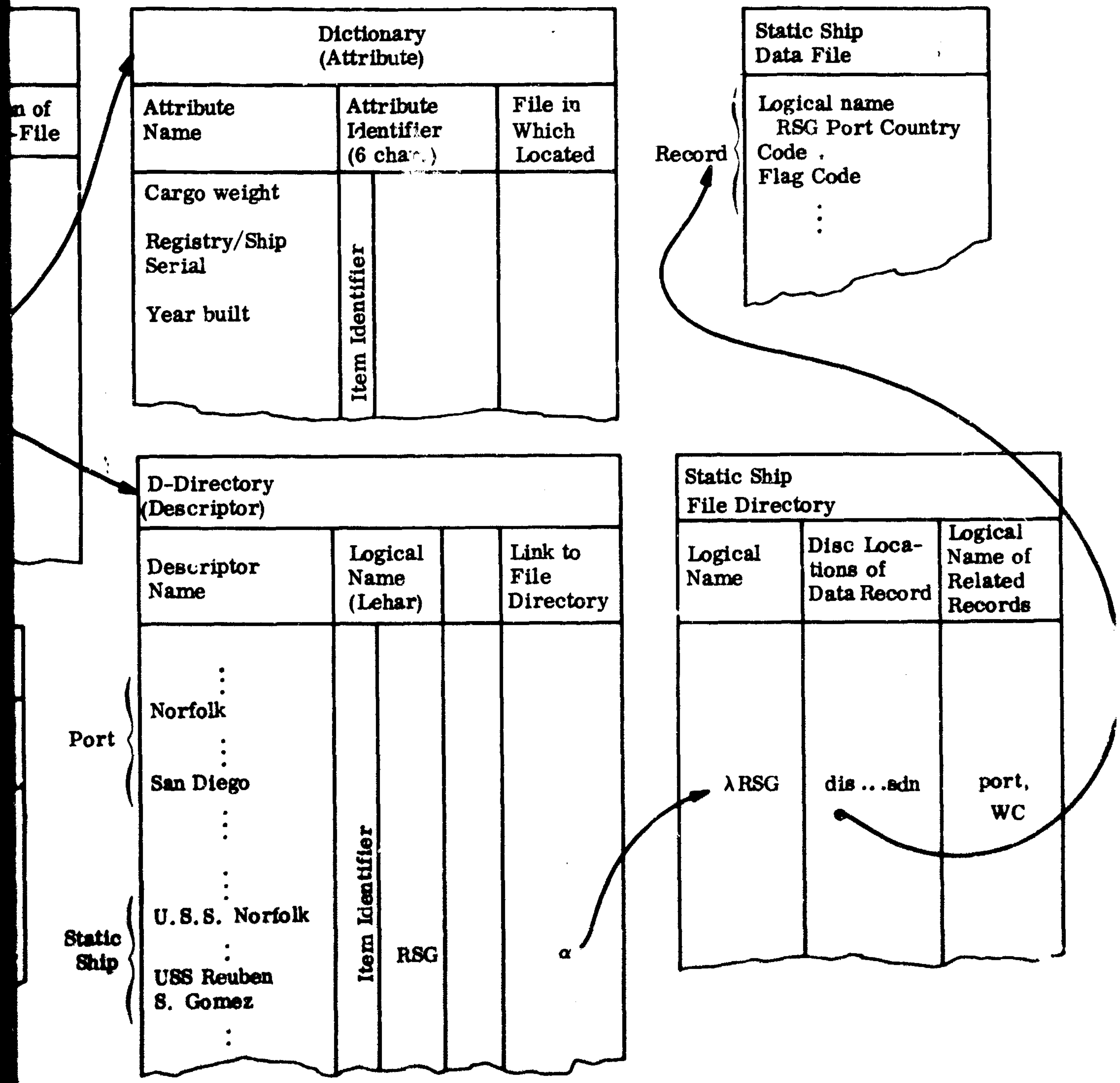
Attribute Name	Attribute Identifier (6 char.)
Cargo weight	Item Identifier
Registry/Ship Serial	
Year built	

D-Directory (Descriptor)

Descriptor Name	Logical Name (Lehar)	
⋮	Item Identifier	
Norfolk		
⋮		
San Diego		
⋮		
U.S.S. Norfolk		RSG
⋮		
USS Reuben S. Gomez		
⋮		

Space Available File

Position Referenced	Number of Blocks Available	Bit Configuration Indicating
		Block status (available or unavailable)



2

Figure 4-1. File Organization for th
Surveillance Problem

specified in a record:

(1) Port Record

Port Name Code (logical name of record)
Port Name
Port Position
Port Country Code
Minimum Approach Depth
Oceanographic Index Number

(2) Personality Record

Name (logical name of record)

(3) Ship (Static)

Registry/Ship Serial Number (logical name)
Port Country Code
Flag Code
Preferred Name
IRCS
Ship Type Codes
Pendant Number
Upright Sequence
Administrative Control
Country Code (owner)
Country Code (lessee)
Country Code (builder)
Year Built
Data Last Major Overhaul
Home Port Code
Name Shipping Line

(4) Cargo Characteristics

Gross Weight
Dead Weight
Cargo Weight
Refrigeration Capacity
Bale Cubic Capacity
Measurement Tons
Number of Hatches
Length Largest Hatch
Width Largest Hatch
Number of Booms
Maximum Boom Capacity

Liquid Capacity
Number of Transfer Stations
Transfer Rate Under Way
Discharge Rate in Port

(5) Operating Characteristics

Function Code
Activity Code
Doctor on Board
Refuel at Sea Capability
Special Signal Facilities
Meteorological Facilities
Strengthened for Ice
Salvage Capability
Washdown Capability
Special Radio Equipment
Operator Watch Period
Navigational Radar
Special Navigational Aides
Length in Feet
Beam in Feet
Passengers Normally
Passengers Emergency
Passengers No Choice
Draft (Maximum)
Communications:
  Operator Period
  Broadcast Guard
  DTG of Shift

(6) Propulsion Characteristics

Propulsion Units
Type of Propulsion
Fuel
Bunker Capacity
Fuel Consumption
Endurance
Maximum Speed
Maximum Shaft RPM
Normal Cruising Speed
Normal Shaft RPM
Blank
Reduction Gearing
Number of Screws
Number Blades Screw
Registered Gross Tonnage
Maritime Administration
Short Description

(7) Weapons Characteristics

ECM Capabilities
Helicopter Capabilities
ASW Sensors
ASROC Capability
Major Gun System
Missle System
Torpedo System
Target Designation/Fire Control

(8) Ship (Dynamic)

Latitude-Longitude Coordinates (logical name)
SOA at this time
Bearing
Wind Velocity
Wind Direction
Barometric Pressure
Relative Change
Weather
Reference Fix Position
Fix-to-Ship Bearing
Visibility
Air Temperature
Ship Movement
Task Designation
Chop DTG
Next Task Designation
Operational Commander
National Number
SIOP Unit Designation
Readiness and Status

4.3.6 Directories

Part of the actual data is implicit in the additional data used to manipulate the data files. The set of additional data is given in what will be termed directories. Directories exist on file, format, record, and data levels.

4.3.6.1 File Location Directory. The file location directory is utilized to find the storage location of all files and directories within the system. The File Location Directory consists of as many entries as there are files. Each entry consists of elements to indicate the disc locations for the data files (Paragraph 4.3.4), file directories (Paragraph 4.3.6.4), and the D-directory (Paragraph 4.3.6.3). The location of the directory is also contained in the file location directory.

4.3.6.2 Dictionary. The dictionary indicates the list of attributes for each record for each file type. Each entry in the dictionary consists of two elements. The first element, consisting of six characters, indicates the attribute's identifier. The first character is an item identifier. If the descriptor is an item (a logical name), this character indicates this fact and conversely. Included in the attribute identifier is a code which specifies the class (alphameric, numeric, binary, etc.) of the descriptor.

Additionally, if the attribute is contained within one descriptor word (along with other descriptors), the last two characters indicate whether the descriptor is a character or bit, as well as giving the character or bit position. The second element of each entry indicates the file that the particular attribute is found in. There exist as many entries in the dictionary as there are attributes in the system. If it is desired to know a record's content for a particular file, the dictionary is searched for all entries whose second element equals the desired file's identifier. The first element of these entries lists the set of attributes found in a record of the particular file. An important and valid assumption is that all records of a data file are identical with respect to possible content (homogeneous file), and that an attribute occurs in one and only one file.

4.3.6.3 D-Directory. The D-directory functions in several ways:

(1) It converts input form descriptors which function as logical names into one-word (48-bit) entries.

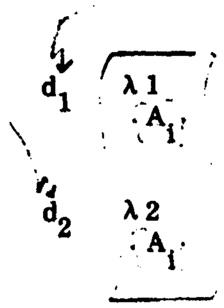
(2) For each such logical name, it lists the total number of data records related to the particular record identified by that logical name. The number of data records referenced is used in the process of retrieval.

(3) It gives the file directory's address to find the list of related logical names.

Because many ships might be reported to the system whose names are not known, even though reliable information is available, provision must be made for ships for which no meaningful normal D-directory entry can be made. In the cases of such "unknown" ships, D-directory entries indicating an external name of "ZZZZZZZZ" are entered for each such ship. In these cases, the routines must be programmed to provide for the situation where the one extended name has many internal logical names associated with it. In all other cases, the relationship between extended and internal names is one to one.

4.3.6.4 File Directories. The file directories are the major data type used in manipulating the actual sea surveillance data. A file directory exists for every data file, and consists of a set of logical names. These logical names may be divided into homogeneous subsets or logical name records. Each logical name record corresponds to one data record of the data file. The logical names of both the data record and the logical name record are the same. The other logical names contained in the logical name record relate the associated records in other data. Associated with the logical name for the particular file is the physical address of the associated record. For example:

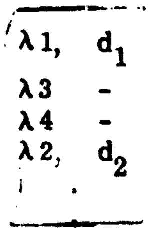
physical record locations on disc



record 1; λ1, logical name of record followed by $\{A_i\}_1$, the set of descriptors.

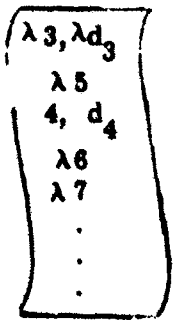
record 2; λ2, logical name of record followed by descriptors: $\{A_i\}_2$

PORT FILE



logical name record for λ1 found in disc position d_i
λ 3 and λ4 are logical names of related records.
logical name record for λ_2 found on physical disc position d_2.

PORT FILE DIRECTORY



λ3 and λ4 are ship's logical names
d_3, d_4 are the physical location of the ship's records

λ5, λ6, and λ7 are logical names of records related to the indicated ship's records.

STATIC SHIP
FILE DIRECTORY

Cross references between data records from any file are accomplished on the directory level. The physical location for any data record from a particular file is given only in that record of logical names of the particular file directory where the logical names are identical. Thus, data is retrieved only through the file directory related to that data.

Some records from one file may or may not reference records from another file. In this respect, the file directories contain these logical names in the indicated record format.

(1) Static Ship File Directory

Registry/Ship Serial Number (logical name)
Disc Location of Data Record*
Logical Names of Related Records:
(a) Port — if in port or Dynamic Ship, if not in port
(b) Personalities
(c) Weapons Characteristics
(d) Cargo Characteristics
(e) Propulsion Characteristics
(f) Operating Characteristics

(2) Dynamic Ship File Directory

Latitude-Longitude (logical name)
Disc Location of Data Record*
Logical Names of Related Records:
(a) Registry/Ship Serial Number

(3) Port File Directory

Port Name Code (logical name)
Disc Location of Data Records*
Logical Names of All Related Records:
(a) Registry/Ship Serial Numbers

(4) Personality File Directory

Name (logical name of record)
Disc Location of Data Record**
Logical Names of All Related Records:
(a) Registry/Ship Serial Numbers

* As many addresses as needed, listed contiguously.
** Actually, none required.

(5) Weapons, Cargo, Operational, and Propulsion Characteristics, File Directories

Logical Name*
Disc Location of Data Record**
Logical Name of All Related Records:

(a) Registry/Ship Serial Numbers

4.3.6.5 Discussion. Each of the eight*** data files consists of a set of homogeneous records. A particular record in a data file is "named" by its logical name and contains a list of descriptors which describe the item identified by the logical name. Every descriptor in the record is given as two elements. The first element identifies the attribute (and denotes those attributes which are items) and the second element is the actual descriptor. Every attribute label as given in the dictionary is one 48-bit word. Every descriptor is given in its basic input form. As many 48-bit words as required are allocated to the descriptor. Every logical name requires one and only one 48-bit word.

The particular descriptors of a particular record may be given in any order with these exceptions:

(1) The first entry in the record must be the logical name.

(2) All descriptors between two logical names must be associated with the first logical name.

(3) A second occurrence (within a record) of a descriptor will be interpreted as historical data.

The file directories require a logical name for each record in the data files. On the premise that many ships have the same characteristics (cargo, propulsion, etc.) it was decided to create records about these characteristics. Thus, the size of the static ship's data file is considerably reduced. But a new problem arises: What is the logical name of these characteristics' records? The procedures used in the data file in grouping attributes into one attribute and the corresponding descriptors into one descriptor word will be used to generate a logical name.

* May be generated as given subsequently.

** As many addresses as needed, listed contiguously.

*** Really, only seven are required since the personality file is implicit in the directories.

4.3.6.6 Space Available File. In addition to the data files and directories, a file termed the Space Available file is needed for bookkeeping purposes. When a file is either maintained or loaded, it is necessary to know where available space exists on disc. The file location directory assists in this process, but is not in itself adequate. The file location directory indicates the range of addresses on disc of a particular file or directory. In the case of data records, which can be stored at random anywhere within the range allocated to the file, it is important to know precisely where the records are stored and if sufficient contiguous space exists for the record. It is desirable to store a data record in contiguous disc locations to speed up the storage and the retrieval of data. As noted in the following paragraphs, which describe the physical file description, an AM will be exceedingly useful to manipulate the space available file.

4.4 PHYSICAL RECORD LAYOUT

A data record in any of the files contains the logical name of the record and a list of attribute-descriptor pairs. The logical name uniquely identifies the record within the file. It is one computer word in length (48 bits), and is always the first word of a record. By virtue of the convention that no more than one record can occur in a physical disc record, it is also the first word in the physical record; when a record is continued over several disc locations, the logical name is repeated in continuation records. An attribute-descriptor pair contains an attribute identifier and a descriptor.

4.4.1 Attribute-Descriptor Pair

An attribute-descriptor pair contains an attribute identifier and a descriptor. Attribute identifiers are one computer word in length and contain a unique identification of the attribute, together with other information about the attribute and descriptor. The attribute identifier is presented analytically in Table 4-1.

Some descriptors are of standard form and fixed length. An example of such a descriptor is a part country code. Other descriptors are inherently of variable length, and are likely to be presented in different forms to the file. An example of such a descriptor is the short description of propulsion characteristics. In order that the sea surveillance data base provide a reasonable representation of intelligence data bases, a provision has been made in designing the data base for a preponderance of variable length descriptors, even though most descriptors could be standardized in a fixed form.

TABLE 4-1. ATTRIBUTE IDENTIFIER

	Field Description	Number of Bits	Bit Location
1.	Attribute Flag This flag is set to 1 to indicate that the word is an attribute identifier.	1	b_{47}
2.	Code Indicator This field indicates the code in which the descriptor is presently stored, e.g., Gray, Hollerith, binary...	5	$b_{42} - b_{46}$
3.	Item Flag This flag indicates whether or not the attribute is an item (and hence, whether it is indexed in the corresponding directory).	1	b_{41}
4.	File ID This field identifies the file to which the attribute belongs.	8	$b_{33} - b_{40}$
5.	Unique ID This field uniquely identifies the attribute.	12	$b_{21} - b_{32}$
6.	Related File Indicator This field indicates which file may also be affected by a maintenance operation performed on the descriptor of this attribute-descriptor pair. For example, if a maintenance operation indicates that a ship has entered port by an operation on the attribute-descriptor pair, the related file indicator will reference the dynamic ship file since the dynamic ship file record for the same ship should no longer reflect a bearing or speed and since the ship is now in port.	8	$b_{13} - b_{20}$
7.	Descriptor Characterization	5	$b_{8} - b_{12}$
8.	Number of Characters for the Descriptor	8	$b_{0} - b_{7}$

4.4.1.1 Attribute Identifier. The major data needed to characterize a data descriptor is:

(1) Code form of descriptor (e. g., Hollerith, binary).

(2) Whether the descriptor is an item.

(3) What file contains the attribute-descriptor pair.

(4) What file contains related data.

(5) Descriptor length.

This data is carried in the attribute identifier. Other data about the descriptor is also carried in the attribute identifier, rather than in internal tables, simply because the space is available:

(1) Data Class: Is the descriptor numeric or alphanumeric?

(2) Data Usage: Is the descriptor primarily used for computational or display purposes?

(3) Format: If the data is numeric, is the field in fixed point or floating point form?

(4) Length Convention: Is the descriptor fixed or variable length?

(5) Justification: Is the field left or right justified?

This data is contained in the descriptor characterization field of the attribute identifier (Field 7, Table 4-1). This five-bit field has the following interpretation:

A B C D E

where A = 1 if the descriptor is class numeric, and
0 if the descriptor is alphanumeric.

B = 1 if the descriptor contains data whose usage is computational, and
0 if the usage is display.

C = 1 if the data is fixed point, and
0 if the data is floating point.

D = 1 if the descriptor is a fixed length field, and
0 if the descriptor is a variable length field.

E = 1 if the descriptor is left justified, and
0 if the descriptor is right justified.

4.4.1.2 Attribute Classification. The attributes of the various data files can be classified according to the information contained in the attribute identifier fields presented in Table 4-1. This classification will sufficiently define the formal nature of the descriptors contained in the attribute-descriptor pairs. The classification of attributes of the sea surveillance data base is presented in Paragraph 4.3.5.

4.4.2 Descriptor Conventions

Alphanumeric and numeric display descriptors will be left justified and therefore begin a word. Since the next entry after a descriptor in a data record must be an attribute identifier, such a descriptor will always occupy an integral number of computer words. Descriptors containing numeric computational data will occupy at most one word in either standard fixed or floating point format. Filler for alphanumeric and numeric display fields will be binary zeros.

4.4.3 Disc Record Storage Conventions

Detailed disc record conventions are given in Paragraph 4.5. For the purpose of this discussion, however, it must be noted that every data record begins a new disc record, and that the first word of each such record is always a logical name.

Because destruction of a directory entry could result in the loss of indexing to trailer records of a data record, each trailer record begins with a logical name, making it theoretically possible to recover trailer records in the event that the corresponding directory entry is destroyed. Hence, trailer records have the same format and the same disc storage conventions that apply to the initial data record.

4.4.4 Dictionary Conventions

The first word of the dictionary will be called the dictionary "NENT" word. It will contain, in the upper portion, the number of disc data records currently storing the dictionary. In the lower portion it will contain the currently correct total number of attributes in the system.

The attribute identifier, as defined in Table 4-1, will always contain binary "ones" in bit positions 0 through 12. The purpose of this convention is to allow differentiation of this attribute identifier from the other element of the dictionary, the attribute name.

The attribute name is the external r presentation, the attribute name, can stand in a many-one relationship with the attribute identifier.

Essentially, the dictionary is a list of attribute-name attribute-identifier pairs. Because the file containing the attribute is identified by a part of the attribute identifier, it is sometimes convenient to think of the dictionary as a list of triplets: attribute name, attribut identifier, file identifier. In this discussion, the attribute-name attribute-identifier pair representation will be employed.

Within each attribute-name attribute-identifier pair, the attribute name appears first, left justified, with filler of binary zeros, if required. Attribute names can be of varying length, and are followed by their corresponding attribute identifier. Since there can be many different attribute names for the same attribute identifier, this implies that the same attribute identifier can occur many times within different attribute-name/attribute-identifier pairs.

4.4.5 Directory Conventions

There are three different classes of directories, each of which is treated separately. The file directories compose one such class; the D-directory and the file location directory are sufficiently different to deserve treatment as separate classes.

4.4.5.1 File Location Directory. The file location directory consists of a unit of file name file address pairs, preceded by a NENT word. The NENT word contains, in the upper portic the number of disc data records currently storing the file location directory; in the lower portion the current total number of files is stated. It should be noted that, for the present sea surveillance system, the file location directory does not require more than one disc data record for storage, and is, in fact, stored in core during the processing of parts of the system such as file maintenance, which must frequently refer to it.

The file name-file address pairs are packed in one word items, with the following format

File ID	bits 24-47
Address Control Word	bits 0-23

The address control word is in the format of the CDC 1619 Magnetic Disc File Controller Reference Manual, pages 3-14, and constitutes the actual disc address of the first record in the file.

It should be noted that, for purposes of the file location directory, all directories, the dictionary, and the data files are construed as files, and are therefore referenced by the file location directory.

4.4.5.2 The D-Directory. The D-directory is basically a list of four-tuples containing the following:

(1) The external-logical name.

(2) The logical name of a record.

(3) The directory address for the corresponding logical name.

(4) The number of data records to which the logical name is related.

The first word in the D-directory is the NENT word, containing the number of disc records currently containing the D-directory in the upper half, and the total number of D-directory entries in the lower half. All subsequent words consist of four-tuples of the stated kind.

The external logical name is a name by which the data is known, and input to the system. It can exceed one computer word in length (unlike the logical name), and therefore must be converted to the logical name. Further, an item, such as a ship, can be known by several names. "S.S. Reben S. Gomez" and "TS#0417" can both be meaningful names of the same object. Hence, there is a many-one relationship between external logical names and logical names. As in the case of the dictionary, separate four-tuples are created for each external name. The external logical name is left justified and zero filled, since it is always either alphanumeric or numeric display.

The logical name is a one-word entry, defined in detail in Paragraph 4.4.6. The directory address is given in the 1619 Disc Controller format.

The number of data records to which the logical name is related is computed as the count (tally) of all disc records required, initial records plus trailer records, for each record presenting information on the named object. This number is also called the norm, and is primarily of value in loading data for the query system. In this system it serves the purpose of optimizing core assignment and disc scheduling.

4.4.5.3 File Directories. File directories are adequately described in Paragraph 4.3.6.4, as long as the following physical restrictions are kept in mind.

(1) The logical names are one-word entries, as defined in Paragraph 4.4.6.

(2) Disc locations are given in the 1619 Controller format.

(3) The first word of each directory is a NENT word, containing the number of entries.

(4) Gaps may occur in the directories as a result of maintenance, in which case the gaps are binary zeroes.

(5) Logical name sequence is maintained except within physical disc records. Hence, if a given physical record does not contain logical name λ, but contains λ' lower in collating sequence than λ, and λ'' higher in collating sequence, the logical name λ is not in the Directory.

(6) Data respecting one logical name does not overlap physical records without repeating the logical name.

4.4.6 Logical Name Conventions

Logical names occupy one computer word (48 bits) and must uniquely identify a record within the system. The most convenient assigned logical names are arbitrarily assigned sequentially ascending numbers. In this system, logical names are so generated, with the additional requirement that 47 equalizers and bits 30 through 46 are always binary ones.

A special case exists for the logical name of records of the dynamic ship file. The logical name of these records is the coordinates of the ship in question. These coordinates are expressed according to the following schema, where each capital letter represents an octal digit:

37777 LAAAAATBBBB

where L = 1 if the longitude is East
0 if the longitude is West

AAAAA = longitude, in radians, scaled 2^{14}

T = 1 if the latitude is North
0 if the latitude is South

BBBB = latitude, in radians, scaled 2^{14}

4.4.7 Space Available File

Experience in assessing the required maintenance operation has indicated that when a file is maintained, the available space within the disc file made available must be noted. Since the file locator exists, it is not complicated to know the pertinent position range for a particular file or directory. Therefore, the space available file is on a disc file level. This information is maintained in the first position of the first disc. Each disc position of the 16 discs (1,024 positions) is represented by four contiguous 48-bit words. $(4i + 1)$; $(a \leq i \leq 7)$ is $(8k + i - 7)/64 = D + P/64$, where D is the disc addressed, P is the position, and $P/64 \leq 1$. The first word of the four-word element associated with a position indicates the number of available blocks at the position (≤ 128). The next three words are construed as one word of 144 bits where the ith bit is equal to 1 if, and only if, the ith block contains data. Each word has an identifier: 1, 2, 3, or 4.

4.5 INFLUENCE OF ASSOCIATIVE MEMORIES ON DATA BASE ORGANIZATION

4.5.1 Introduction

In considering the organization of data for an associative memory, one must consider the time to transfer the data from core to associative memory (as in the case of GAP), and the various aspects of processing: interrogations of the data base, maintenance orders to change the data base, and new data inputs to be stored in the data base. All of these factors affect the overall system and are considered separately in the sections of this volume which discuss these subjects (Sections V, VI, and VII). It is important to discuss the logical considerations relevant to the structuring of the data which are influenced by the characteristics of an associative memory. This paragraph only describes the logical considerations, does not take a total system view, and does not provide timing information.

4.5.2 Data Organization Considerations

As noted previously, the indexing scheme utilized in this study is a multi-list file structure in which the linkage to records is extracted from data records and placed in descriptor lists. From a timing consideration this organization of data achieves a faster time to retrieve data records than that of a multi-list file structure with linkages in data records. Maintenance of data records is equally cumbersome in both systems. This pertains whether or not an AM is utilized. The system is input-output bound and the AM has no effect upon this. However, the logical organization of data within the individual files has changed significantly as noted in the following paragraphs.

4.5.2.1 Data Record Organization. As described in Paragraph 4.4, a particular data record consists of a logical name of the record, and a list of attribute-descriptor pairs. No sequencing of attributes has been defined; with an AM, no sequencing is required.

If no AM were available, the above organization would require that each attribute be scanned until the proper one was found. By imposing an ordering on the data (say alphanumeric), the average time to find the attribute could be decreased.

Because of the logical capability of an AM, the ordering of attributes does not enter since the time to retrieve the descriptor in an AM is independent of its relative location with respect to other attributes. Not having to be concerned with sequencing of attributes is thus due to the logic of the AM. For a variable size record with variable size fields, as is the sea surveillance data base studied in this report, this is particularly important. However, in processing a single fixed field size with fixed record size format, an AM is logically no more powerful than core. As is the normal case with such records, a data description exists so that one can go directly to the desired descriptor once the starting location of the record is known. Since all data records considered for the sea surveillance problem are smaller in size than the AM considered (2,048 data words), the entire data record and, indeed, several records may be stored and processed simultaneously in an AM thereby enhancing the AM over core. The ability to process data records in parallel whether fixed field, fixed size, or variable field makes the AM a powerful processing tool. This advantage is achieved in the A3 because,in many cases, no data transfer time is required. However, in GAP it is advantageous only if a sufficient number of searches is applied to the data records. For example, note that in the case of the REL subroutine, which retrieves logical names of records given an attribute-descriptor pair on which one desires to retrieve, it takes 7.3 milliseconds utilizing GAP and considering data transfer time, and 5.9 milliseconds without an AM.

4.5.2.2 Directory Organization. The discussion in Paragraph 4.5.2.1 also applies to the directories. In particular, the influence of the AM on the dictionary, on the directory for an attribute, and on the D-directory is discussed. Similar considerations apply to the other directories.

(1) Dictionary. The dictionary lists for each file all attributes, their machine representation, and the description of the data. Storing the dictionary in the AM when it is required has the same utility as a data record.

(2) D-Directory. The D-directory contains a list of logical names of records. A D-directory record contains the number of data records containing the descriptor and the address of the list of addresses of the data records containing the descriptor. As noted previously, the D-directory is divided into K segments where any segment may be placed entirely within the AM. Within segments, there is no ordering of directory records. However, between the i th and i + 1 th segment, the directory records in the i th segment precede those in the i + 1 th segment. (For example, in the SHIP ID D-directory, all ship IDs listed in the i th segment are alphanumerically less than all ship types listed in the i + 1 th segment.

If no AM existed, the items in the i th segment would be sequenced according to some criterion. If the list of descriptors is large, the necessity for sequencing arises so that one does not have to sequence through the D-directory to get at the descriptor. Rather, one can go to a small directory in the D-directory, find the segment in which the descriptor appears, and go to the appropriate segment directory. Thus, the AM did not change this concept, but merely simplified maintenance within segments and the process of retrieving the appropriate directory record from amongst the set of directory records in the segment. Maintenance of the D-directory is vastly simplified because of the AM and could have a substantial amount of coding. Provided that one ignores transfer time to the AM (which is not necessary for memory A3, but is necessary for the GAP, A1 and A2 memories), maintenance time for segments and retrieval time for descriptors in segments are faster in the AM than in core.

(3) Directory. The directory consists, essentially, of the list of logical names of records which contain the descriptor. If processing were accomplished in core, for efficiency the list of logical names of records would be ordered. However, if the directory record is to be processed in AM, the logical names need not be ordered.

4.5.3 Disc Allocation

An essential element of the data organization is the requirement for the available space on disc where data can be stored. If no AM exists, then several possibilities exist:

(1) A directory can exist which contains the names of those disc positions which have a specified number of available blocks. Within two disc accesses one can determine a position which has the specified number of available blocks. Finding available space is not time consuming, but maintaining the directory is time consuming when one has to take a position off one list and place it on another list. Problems also arise when several positions must be utilized to find the appropriate amount of space.

(2) Instead of maintaining a directory, a word in a position could be reserved to specify the amount of available space in the position and could give a link to the next position with the same available space. This solution is equivalent to (1).

The method proposed in this study is logically more powerful than either of the above approaches. By loading into AM a map of the available and unavailable disc locations allocated to a record for the particular file desired to be stored, and the number of blocks available in each position, the determination of where available disc space is is simplified. If necessary, one can determine those positions which have a specified number of contiguous available blocks in a disc position. Thus, greater flexibility is achieved.

Logically, in maintenance one knows the position and block that have become available. Hence, if the disc map is in core, one can go there directly rather than utilize the AM for this function. However, maintenance is simplified substantially because it is not necessary to change the list on which the maintenance position belongs.

Although it appears that the AM has an advantage in this instance, it is only illusory. It is not practical in this problem to have the space available file reside permanently in the AM. Transferring the file back and forth between core and AM will be more time consuming than performing core operations. This applies equally to A3 and to GAP. In maintenance operations, the space available file will be required permanently in non-AM core and transfer time to either GAP or A3 would be too costly in time. An advantage would accrue, however, if the space available file and other tables required by an Executive Control Program were to reside permanently in an AM.

4.6 IMPACT OF DISC FILE ON ASSOCIATIVE MEMORIES

The main objective of this study is to assess associative memories used in an operating system whose configuration includes a disc file. In all configurations considered, except that one which includes the A3 memory, data is read into (or from) the associative memories from (or into) 1604-B core memory. Thus, in these configurations, the topology of data stored on a disc file is independent of the associative memory. In like manner, the minimum number of words read into 1604-B core memory from a disc file (i.e., a block) is independent of the AM since another data transfer operation is required to place the data in the AM. The number of words transferred by the second data transfer operation may be independent of the block size.

It is possible in the A3 memory configuration, to transfer data directly from the disc into the associative memory, and conversely. In this case, it is important that the A3 available memory be at least equal to a block size.

The preceding paragraph points out the fact that there is no direct impact of the disc file on the associative memories; however, there may be an indirect time dependent impact. Perhaps this point is illustrated by the fact that the average access time for disc data is 225 milliseconds followed by an average word read rate of 90 microseconds. If one block is "randomly" read from the disc file, an average time requirement is (225 ms + 32 × 90 microseconds) = 227.88 milliseconds. In this time it is possible to transfer the equivalent of one 32K 1604-B core memory to the associative memory.

In much of the processing that is to be accomplished in the sea surveillance problem, the data disc addresses that must be retrieved for subsequent processing are often known, a priori. In this case, it is impossible, depending on available memory space and processing order of the data, to order these disc addresses such that the disc access times are minimized. The objective of this reordering is to minimize the impact (or bias) of the disc file upon the results of this study. The routines that perform this optimal reordering and read disc data into available 1604-B core memory are the query preprocessor and the controller.

Whenever timing is considered with respect to an associative memory, the dependence between disc input-output operations and performing associative memory transfers becomes of interest. Whenever associative memory operations are performed in the "burst" mode, disc operations are interlocked. However, if the "buffer" mode is used, it is possible to saturate the 1604-B with a resulting loss of data. This saturation must be prevented by software techniques.

SECTION V. INPUT MESSAGE PROCESSING

5.1 INTRODUCTION

The sea surveillance data base is a collection of pertinent information about the current situation regarding all ships, ports, and personalities within the surveillance area. Since the real life situation is dynamic, the representing data base is also dynamic. If a "snapshot" is taken of the real world and the result considered as an initial data base, then this data base is altered in a dynamic sense by: internal projections (e.g., advancing all ships), maintenance orders (e.g., delete duplicate records), and input messages (e.g., ship X has left port Y).

The task of the input message processing subsystem can be described as one of translating messages about the world into various elementary instructions to the file maintenance subsystem. This section of the report is concerned with input messages. It describes and gives examples of such messages, outlines the required processing, and describes the role of the associative memory configuration in such processing.

5.1.1 Availability and Formats

In actual implementation, a flexible, English-like language would be provided for input messages. This would probably be an extended form of query language. In such a system, the first step of processing a message would be its reduction to a more "machineable" form. Since that kind of process is examined in some depth in other sections (Section VII), it is not developed as part of the input message processor. Rather, input messages are assumed to be in the form of the available test data, that is, 80-character fixed field card images as described in the IBM report entitled Sea Surveillance Data Base Representation as Test Vehicle.

For purposes of this study, the scope of the input messages will be sufficient to

(1) Change Commanding Officer of Ship Serial AM 3572
to CAPT R OWENS.

(2) Position of Ship Serial AM 3572 is 42.4° No '0675° W, Speed Advance is 30, course is 266.

(3) Ship Serial AM 0458, Task Designation XXXXX; Operational Commander is Y - Y (16 characters) Readiness and Status ZZZZ.

(4) Ship Serial AM 3133 has been assigned to the Valley Forge, Type CVS Hull #45, Max. Speed 33.0, Normal Speed 15.8, 4 Screws, Function SUACCAR.

(5) Ship Serial LI0386 has a doctor aboard.

(6) Ship Serial AM 3572 is in Boston, Mass.

(7) Unknown Ship at 273N, 0723W; Speed of Advance 20, Course is 133.

The input message processor will handle these examples, and has been designed and programmed in such a general manner as to accommodate an even greater variety of input message forms. The major criterion for whether an input message about the real world can be processed by the subsystem is whether the message can be reduced to a format called the "input string." The input string consists of a beginning message code, an action code, a set of attribute descriptor pairs, and an end of message code. In general, this criterion can be satisfied if a system routine can exist which will translate input message codes (both explicit and implicit) into action codes and attribute descriptor pairs. Since such routines are easily defined for conventional input messages, their existence is assumed. Hence, the routine developed in this section will show the processing required to break apart general input strings into substrings concerning single items, and the analysis and disposition of each type of substring.

5.2 INPUT MESSAGE – PROCEDURAL LOGIC

5.2.1 Types of Messages

Messages entering the system are concerned with ship movements, ship characteristics, port characteristics, and personalities. The input message processor has as its main objective the association of the input message with the proper family of records contained in the data base. Messages may affect data records and directories (or neither). The disposition of each type of message, in general, uses the following rules.

5.2.1.1 Confirmations. If the message contains a descriptor that matches the associated descriptor, it is a confirmation and no action occurs.

5.2.1.2 Contradictions. If the message contains a descriptor that contradicts the associated descriptor,the following occurs:

(1) If the file is the dynamic file, enter the new value and correct affected di. ectories.

(2) If the file is the static file, flag for management action. If the change is desired, re-enter with a maintenance order.

(3) If the file is the port file, follow (2).

(4) If the file is one of the characteristics files, follow (2).

(5) If the file is the personality file, follow (1).

5.2.1.3 Additions. If the message contains information about a new item, generate a maintenance order to file the item record a..d associated directories. If the logical name of the new item is not known, assign a temporary logical name until its true logical name becomes available. If the message gives additional descriptors for items already in the system, proceed as in Paragraph 5.2.1.2.

5.2.2 Input Message Processor Organization

The general approach for handling input messages is essentially a two-phase process. The first phase, performed by the input processor, is basically an interpretation phase. Its function is to examine the input message, and generate a set of instructions to the file maintenance system. The second phase performed by the file maintenance system enters the data base revisions called for in the list of file maintenance orders presented to it.

The process of identifying items involved in a message, and distributing the information contained therein appropriately may require considerable manipulation. The approach used by the system is to save all intermediate information developed during the message analysis, and pass it down in the form of maintenance order trailers.

Once the input string is available for processing, the input message processor subjects it to three separate routines:

(1) The Attribute Split Routine creates a rectangular array of attributes and descriptors affected by the string, called the message instruction table.

(2) The Sub-String Processor examines the rectangular array, determines what actions may be called for, and reads in needed data from disc.

(3) The Maintenance Order Generator translates the results of prior routines into file maintenance orders for processing by the file maintenance routine.

The organization of these routines is presented in Figure 5-1. Each routine is described in subsequent paragraphs.

5.3 ATTRIBUTE SPLIT ROUTINE

Each message string relates to, at most, one ship. The information carried in an input string may relate to many data base items. For example, the message "ship X is in port Y" relates to a ship item and a port item. In order to

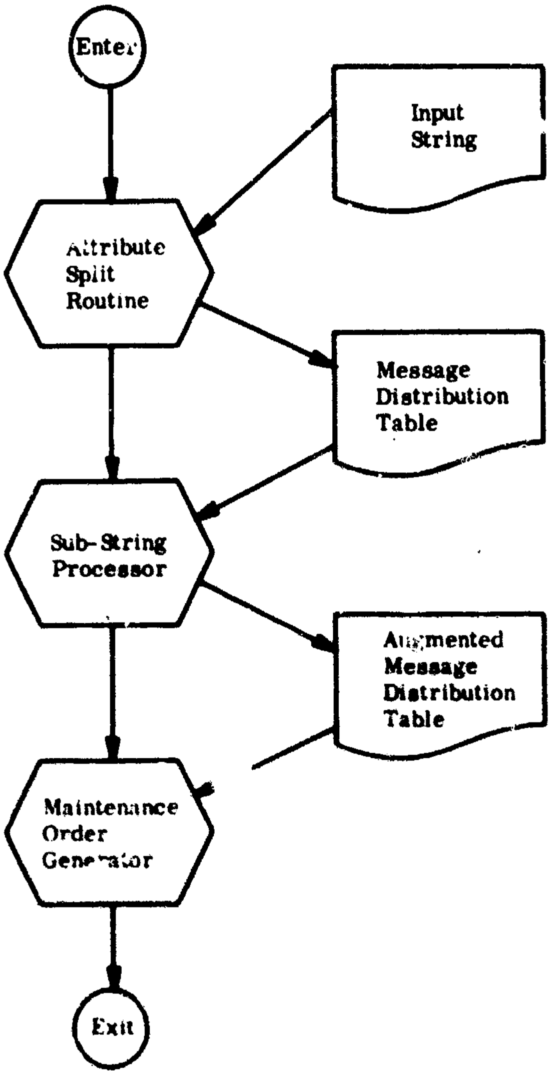

Enter
Input String
Attribute Split Routine
Message Distribution Table
Sub-String Processor
Augmented Message Distribution Table
Maintenance Order Generator
Exit

Figure 5-1. Input Message Processor Organization

process such messages, a table structure is utilized by a routine called the attribute split routine. The table structure is used to contain the data relevant to different items. This table, called the message distribution table (MDT), may be explicit or implicit. That is, attribute-descriptor values may be entered into it, or lists of attribute-descriptors may be labeled with table positions. Which of the two forms is chosen depends on the availability of an associative memory, and on details of file and record organization. In this system the attribute-descriptor representation of data within the record and the availability of an associative memory make it convenient to describe an input message processor emphasizing an explicit message distribution table.

5.3.1 Message Distribution Table (MDT)

The MDT is a rectangular array of attribute-descriptor pairs. Each cell is labeled $E_{i,j}$, where i varies with the file type and j varies with the attribute. The value j = 0 is reserved for logical names of items. The message distribution table is presented schematically in Figure 5-2.

5.3.2 MDT Processing

The Attribute Split Routine clears the MDT, finds all attributes that are logical names (via the D- , enters them for j = 0, i = value appropriate to file (e.g., i = 0 implies ship static file, i = 1 implies ship dynamic file, etc.). For other attributes that are not items, the Attribute Split Routine scans the dictionary to find whether or not they are in the system, and if they are, in which file they are to be found. Each attribute-descriptor pair is then entered in the appropriate column (or labeled with the appropriate i, j value), until the complete message string is exhausted. The MDT is now "loaded" and ready for analysis by the sub-string processor.

5.3.3 Outline of Processing

The associative memory proves very convenient for processing the input string. The processing performed by the Attribute Split Routine can be described as follows, when a GAP-like associative memory is available:

* (1) Clear the associative memory.

* (2) Load the message string into the associative memory.

* (3) Set up mask and comparator for attributes.

* (4) Set responders for all attributes.

* (5) Read addresses of all responders into the list L_1. (If there are no responders, exit, reporting an error condition.)

* (6) Perform a compare for items, and "and" the response store. This step turns on a final response store for logical names. (If there are no responders, exit, reporting an error condition.)

*All steps marked with an asterisk (*) are steps performed by an associative memory. Step (8) requires both associative and conventional processing.

i =	0	1	2
j=	Ships Static	Ships Dynamic	Port Characteristic
0	Ship Code/XXXX	not given	Port/YYYY
1		velocity/0	
2			
3			
4		$E_{i,j}$	

Figure 5-2. Example of Message Distribution Table

* (7) Advance buffers one position. Readout responders into the list L_2.

(mixed) (8) Set up control for the AM dispatcher, and use the dispatcher to load the D-directory in segments into unused parts of the AM. Compare each segment against L_2 for equalities; all D-directory entries that get responders on this comparison are read into the list L_3.

(9) Store the list L_3 for processing by the file maintenance subsystem. This list is a part of the "trailer" described in Section VI.

(10) Determine the current length of the list L_2; call this length M.

(11) Set $j = 1$.

(12) Compare the j^{th} element of L_2 with the j^{th} element of L_3.

(13) If a match is determined, a logical name has been found. Enter it in the MDT; otherwise, go to step (8).

(14) Set j equal to $j + 1$.

(15) If j does not equal M, go to step (12); otherwise, go to step (16).

(16) Set flag "f" in the MDT, and enter a logical name.

(17) Step K by 1.

* (18) Clear D-directory portion of the AM.

* (19) Load dictionary into available AM space.

* (20) Read attributes from dictionary into core and set up comparator register.

* (21) Perform a selected compare for equality on the dictionary, if there is no match, an error has occ:rred; otherwise continue

* (22) Get the file index from the dictionary. This is the i value of $E_{i,j}$.

* (23) Random block unload data to next attribute location. Store attribute in MDT at $E_{i,j}$, where j = j + N for that i.

(24) Repeat steps (20) to (23) until the list L_1 is exhausted.

These steps are flowcharted in Figure 5-3.

The following comments are likely to be helpful in understanding the algorithm and flowchart:

(1) The Attribute Split Routine records the locations of all attributes in the message on list L_1. It then finds those items which are logical names, reading their contents into list L_2.

(2) Next it loads in the D-directory using the AM dispatcher. Any responders cause the entire D-directory entry to be loaded into L_3.

(3) It is necessary for the order of L_2 to be maintained in L_3 because the next operation is performed in core.

(4) The lists L_2 and L_3 are compared to each other to match logical names. In the event that a name cited in the input message is not in the D-directory, flag f is set when the name is entered in the MDT. When the D-directory is no longer needed, the dictionary is loaded into the AM.

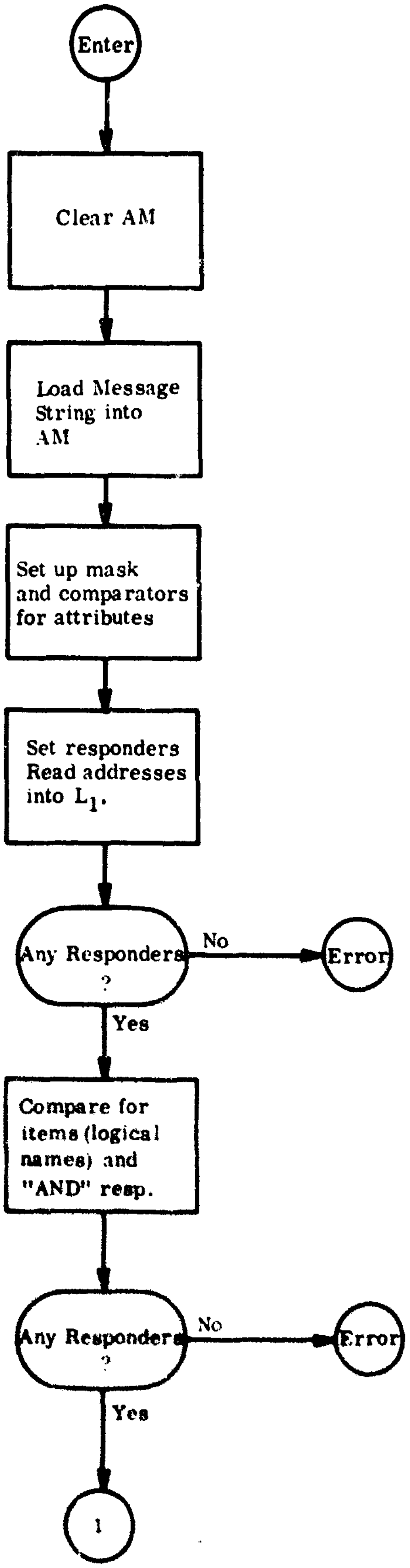

Enter
Clear AM
Load Message String into AM
Set up mask and comparators for attributes
Set responders Read addresses into L_1.
Any Responders ?
No
Error
Yes
Compare for items (logical names) and "AND" resp.
Any Responders ?
No
Error
Yes
1

Figure 5-3. Attribute Split Routine (Sheet 1 of 4)

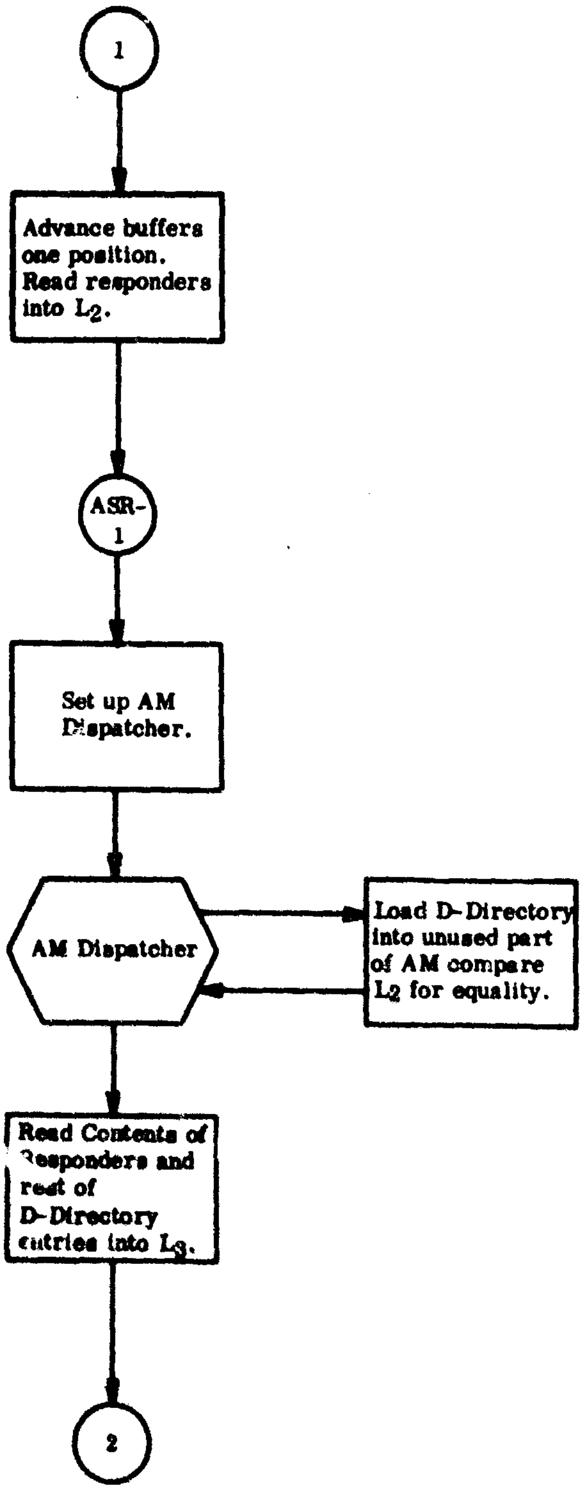

1
Advance buffers one position. Read responders into L_2.
ASR-1
Set up AM Dispatcher.
AM Dispatcher
Load D-Directory into unused part of AM compare L_2 for equality.
Read Contents of Responders and rest of D-Directory entries into L_3.
2

Figure 5-3. Attribute Split Routine (Sheet 2 of 4)

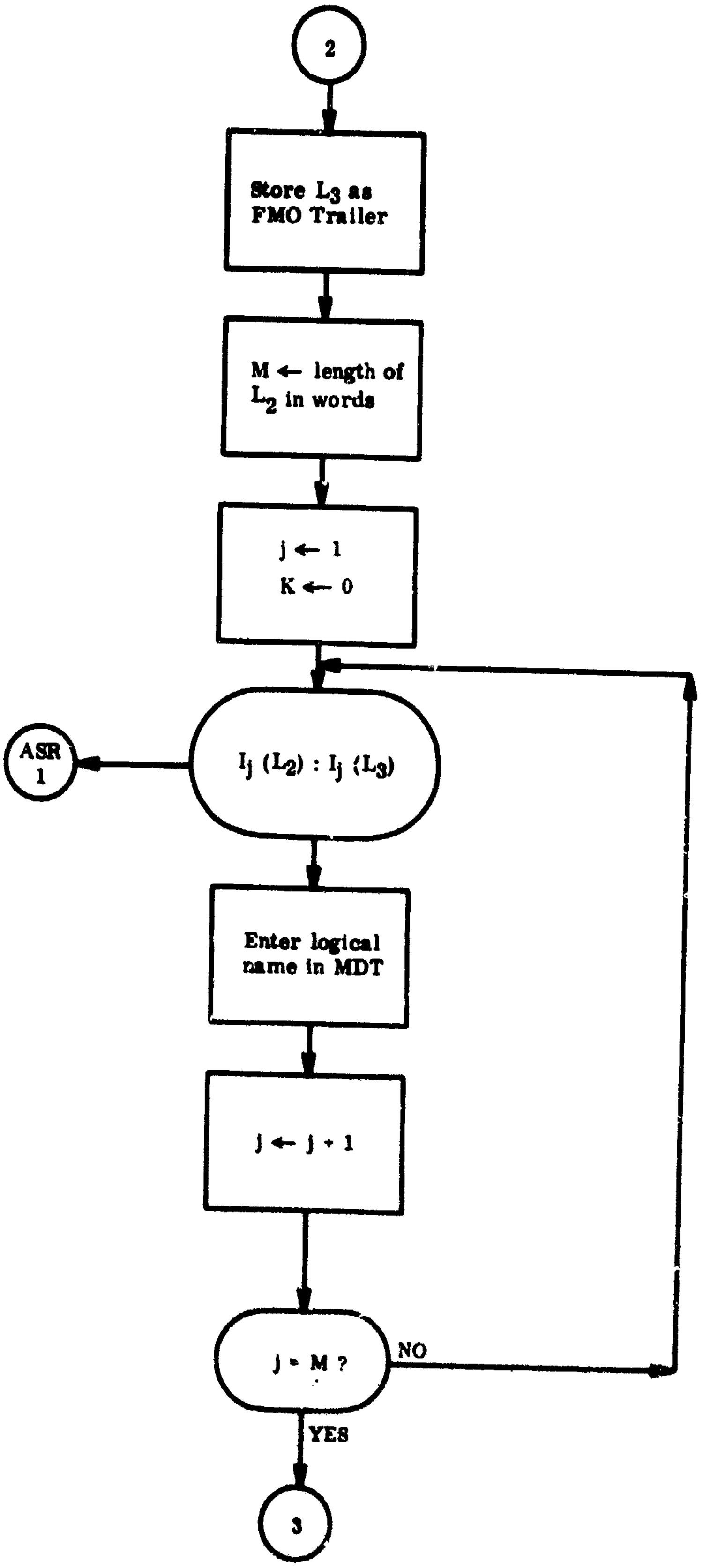


Figure 5-3. Attribute Split Routine (Sheet 3 of 4)

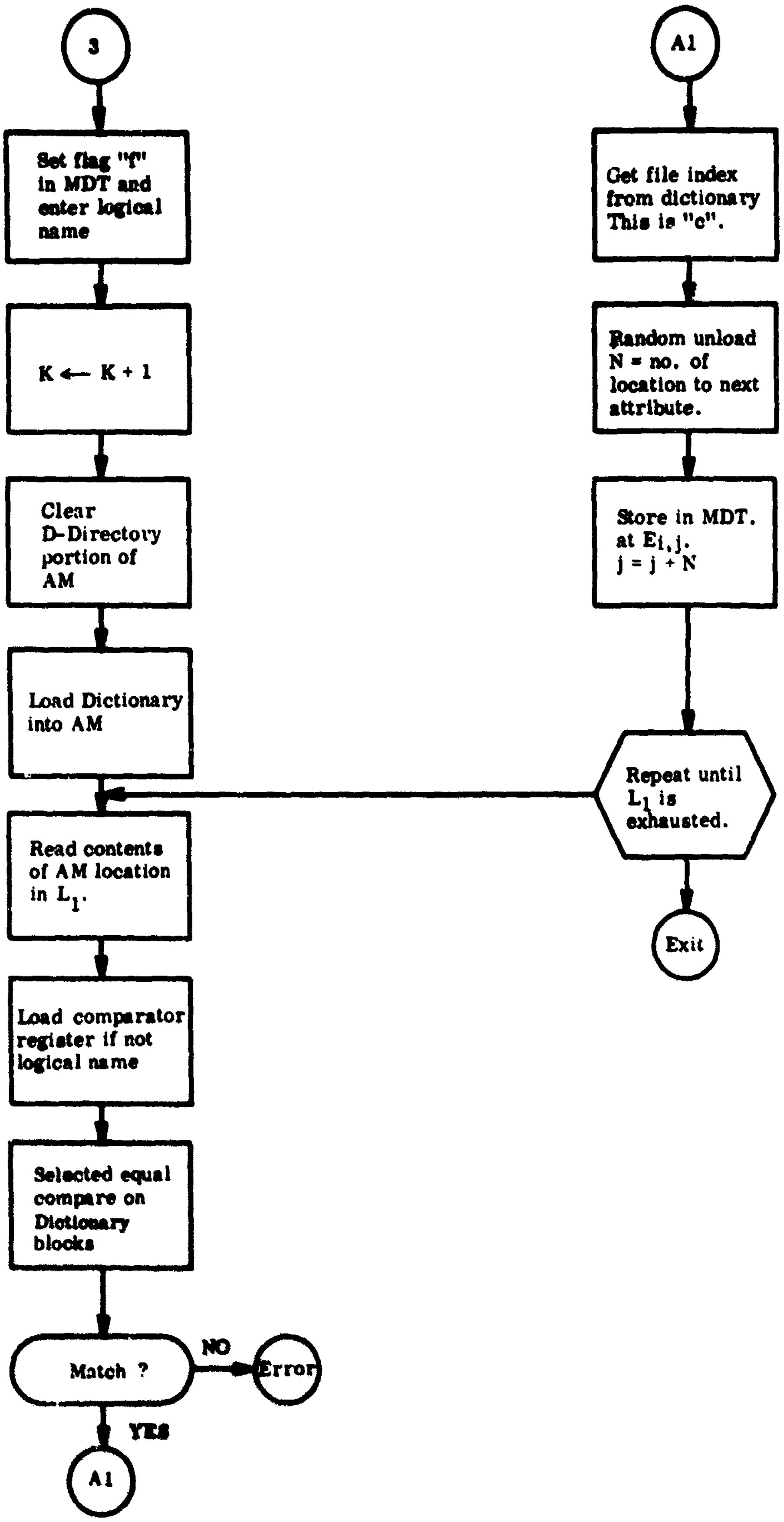

3
Set flag "f" in MDT and enter logical name
K ← K + 1
Clear D-Directory portion of AM
Load Dictionary into AM
Read contents of AM location in L1.
Load comparator register if not logical name
Selected equal compare on Dictionary blocks
Match ?
NO
Error
YES
A1
A1
Get file index from dictionary This is "c".
Random unload N = no. of location to next attribute.
Store in MDT. at Ei,j. j = j + N
Repeat until L1 is exhausted.
Exit

Figure 5-3. Attribute Split Routine (Sheet 4 of 4)

Before drawing conclusions about the applicability of the associative memory for the Attribute Split Routine, it is necessary to state the following assumptions:

(1) It is assumed that the data base is given as in Section IV. Data base designs of a different orientation could invalidate the conclusions. For example, if the attribute descriptor form of representation had not been selected, the routine could be far more complex in the AM.

(2) Certain routines must be programmable for the associative configuration, especially a routine or routines to subset the memory (SPO of Paragraph 5.3.4), and the AM dispatcher (see Paragraph 7.4).

With these reservations in mind, the following general conclusions can be stated.

(1) Programming the Attribute Split Routine is easier using an AM than it would be a conventional memory. Housekeeping steps are kept to a minimum.

(2) The ability to shift and "and" buffers is vital to the stated version of the Attribute Split Routine.

(3) The Attribute Split Routine can be expected to be at least as efficient as a conventional memory version of the same routine. For cases typified by the examples of Paragraph 5.1.1., it is more efficient to use the AM.

The following specific comments are appropriate.

In the analytical phases of the input processor, the AM is convenient for searching directories. In particular, since the input message string occupies a relatively small space, it may be held in AM for the entire Attribute Split Routine, while the remainder of the AM can be loaded with directory data for searches. During this routine, therefore, much use would be made of the SPO (see Paragraph 5.3.4) so that separate searches are applicable to different segments of AM. In addition, the AM dispatcher is called upon to handle the list segmentation where, for example, D-directory information exceeds available AM storage.

A provision that would be convenient in this connection would be the ability to load the contents of a responder into the comparator register without the need to go through a core write and read.

5.3.4 Operations Assumed in the Attribute Split Routine

There are two programming tasks which are assumed in the preceding discussion of the Attribute Split Routine. These tasks are:

(1) Subset the associative memory so that an operation will be performed on only a portion of the data contained in the AM, rather than on all data.

(2) Obtain the next prior or next succeeding responders in the AM (read next responder).

A programmer can, of course, find many ways to perform these tasks, some of which will be more efficient for a particular problem than others. Throughout the entire sea surveillance system, with exceptions made for only a few functions in query processing, standardized macros are employed for these functions with GAP. This paragraph presents these macros, reviews the reasons whey they are used, and draws several conclusions about the GAP hybrid configuration.

5.3.4.1 Selected Perform Operation (SPO). SPO is a macro which performs an operatic on specified subsets of AM locations. The subsets can be defined by stating either addres bounds (beginning GAP address and ending GAP address), or by the contents of response store (subset on responders). SPO therefore has two options called "search responders" and "search-between-addresses", respectively.

(Note: It is possible that setting the LDR before search instructions would eliminate the need for "search between addresses". Although it is stated in the programming manual that an LDR must precede all search instructions, except for MAX and MIN, the exact function of the LDR is not stated under the detailed descriptions of the search instructions. However, from a programming viewpoint, using the SPO routine would be easier since setting up the LDR is quite cumbersome. From a time consideration though, using the LDR would be more efficient.)

Input parameters for search responders option are a specification of the operation(s) to be performed and a value 1 for the parameter n, indicating the search responders operation. For the search-between-addresses option, n is set to 2 and the beginning (θ_1) and ending (θ_2) addresses of the subportion must be specified, as must the operation(s) to be performed.

For the search responders option, SPO proceeds as follows:

(1) Complement buffer, yielding all active non-responders.

(2) Erase responders (sets busy bits = 0 for all non-responders).

(3) Perform the operation(s) specified in input parameters.

(4) Get the results of the operation(s) and hold.

(5) Complement buffer.

(6) Write into responders, using a mask of all zeroes, in order to turn busy bits back on.

For the search-between-addresses option, the following steps are performed:

(1) Perform random block erase, with $R = 0$, $N = \theta_1$.

(2) Perform random block erase, with $R = \theta_2$, $N = 2048 - \theta_2$.

(3) Perform specified operation(s).

(4) Get results of operation(s) and hold.

(5) Activate, $R = 0$, $N = \theta_1$.

(6) Activate, $R = \theta_2$, $N = 2048 - \theta_2$.

5.3.4.2 Read Next Responders (RNR). RNR reads out data associated with a match key where variable formats are used. For example, RNR reads out the logical name of a record chosen by a match on another field. The routine has two options: the "read prior responder" and the "read next after" option.In the "read prior responder" option, the parameter n equals 0. In the "read next after" option, the parameter n equals 1. The parameter A specifies an AM location, normally the location of a responder obtained either by a RFR or by RNR itself. A parameter r is set to 1 if the contents of the AM are to be restored; r equals 0 if the contents need to be restored.

For the "read next after"option, the following steps are performed:

(1) Random block erase, R = 0, N = the parameter A.

(2) Read contents of first responder.

(3) If r = 0, exit.

(4) If r = 1, activate; R = 0, N = the parameter A, and exit.

For the"read prior responder" option, the following steps are performed:

(1) Hold response store in buffers.

(2) Random block erase; R = the parameter A, N = 2048 — the parameter A.

(3) Read count of responders. Call this "C".

(4) Erase first responder. Perform C - 1 times.

(5) Read contents of responder. This is the desired field.

(6) If r 0, exit.

(7) Load buffer into response store.

(8) Write into responders with a mask of zero.

(9) Activate, R the parameter A, N = 2048 — the parameter A.

5.3.4.3 Comments on SPO and RNR. SPO and RNR are frequently used and indicate, by their complexity, the relative difficulty of performing the two operations in GAP. Because of the frequency with which the macros are used, it is strongly recommended that a study be performed for augmenting GAP and GAP-like configurations, with hardware features simplifying such tasks. Such a study will probably reveal numerous alternative hardware solutions.

Throughout all remaining discussion of input message processing and file maintenance, SPO and RNR are assumed. Attention is called to them only where their use is especially worthy of note.

5.3.4.4 "Tag Memory" Version of SPO. While not strictly applicable to the input message processor, owing to the neccessity of using both halves of the 2048-word memory for storing data, a version of SPO exists which is preferable whenever only one half of memory is used as a "tag" memory for the other half.

The procedure applicable to SPO in this approach is as follows. Segment the memory into upper and lower portions, where the lower portion is used only as a "tag" memory containing field search criteria (and/or field search results). Allow 10 bits to contain the address of the words in memory. The object is to use the lower portion of the AM for subsetting the upper portion of AM. An example of the approach shows the ease of using this technique. SPO, for subsetting between addresses, is now much simplified.

(1) Perform a BLC on the 10-bit address portion of the data in the lower memory. This step sets bit response store for all data between the addresses in question.

(2) Copy response store into the E buffer.

(3) Perform the search required on the upper half of memory, and "and" the results with the contents of the buffer.

(4) Read out responders; they are the desired results.

Variations of this approach exist for other subsetting problems; and wherever half of the full GAP storage is required, they can be employed. The tag memory concept is explored in greater detail in Paragraph 7.9.

5.4 SUB-STRING PROCESSOR

This routine examines the message distribution table to determine what actions must be taken. During this process, if it proves necessary to retrieve data records or directories, the pertinent data is held as a "trailer" to the resulting file maintenance orders. (See Paragraph 6.3.7.)

The Sub-String Processor scans the MDT column-by-column, each column relating to a file in the system. Depending upon what is present and what is absent in each column and what action code was given by the input message, the Sub-String Processor generates a set of file maintenance orders. If both the logical name and attribute/descriptor data are in a column, then an A1 or C1 (add data to record or change data in record) will result from that column. If there is only a logical name and no other data, that logical name will in most cases be required for finding the logical name of another column which has attribute/descriptor data. In that event, the Sub-String Processor will pull in the appropriate directory and find the required logical name.

When all family links have been assembled on the MDT, the contents are examined to determine what file maintenance is required. Then, a set of file maintenance orders is generated and control transferred to the file maintenance routine.

5.4.1 Flags Used

In scanning the MDT, the Sub-String Processor looks for certain markers or flags that bear upon the subsequent actions that will be required. One of these the "f" flag which is entered by the Attribute Split Routine when a logical name is cited in a message, but unknown to the system. A second is the "D" flag which is written into each list of the MDT if data other than an item name is given. The most important indicators, however, are the GOBACK indicators and the SUBSEQ flag.

5.4.2 GOBACK Indicator

During the processing of the MDT, if a logical name is missing from a column that will result in an A1 or C1 FMO, any directory entries previously pulled in are examined to see if the link is given. If not, a pushdown stack called GOBACK is generated, with one entry for each column bypassed because of a missing logical name. After all columns have been attempted, the GOBACK stack is processed one entry at a time. If all prior directory lookups have failed to provide a logical name, the system will create one, generate and refer ED FMO, store it in the MDT, and re-enter to process that column in the usual way.

5.4.3 SUBSEQ Flag

For the special case when a position of a given ship is reported, a change in the dynamic file is required. Since lat/long is the logical name of a record, the following procedure will be used to handle this circumstance:

(1) The old lat/long record will have changes made if any are required in the data (e.g., speed of advance course).

(2) The logical name of the record will be changed from old lat/long to new lat/long.

Since the order of maintenance is critical here, a SUBSEQ flag is turned on before the FMO "Reassign logical name" is entered in the list of FMOs, while the C1 is entered in its ordinary place. At the end of the routine, SUBSEQ flag, if on, causes the RA FMO to be appended to the list of FMOs.

5.4.4 Comments on the Sub-String Processor

Quite efficient means exist for performing the tasks of the Sub-String Processor without utilizing an associative memory. The Sub-String Processor is presented in flowchart form in Figure 5-4. No conclusions are relevant other than that no advantage can be seen in using the GAP configuration for the task. However, of the other associative processors, A3 is preferred, owing to its ability to function entirely as a conventional processor.

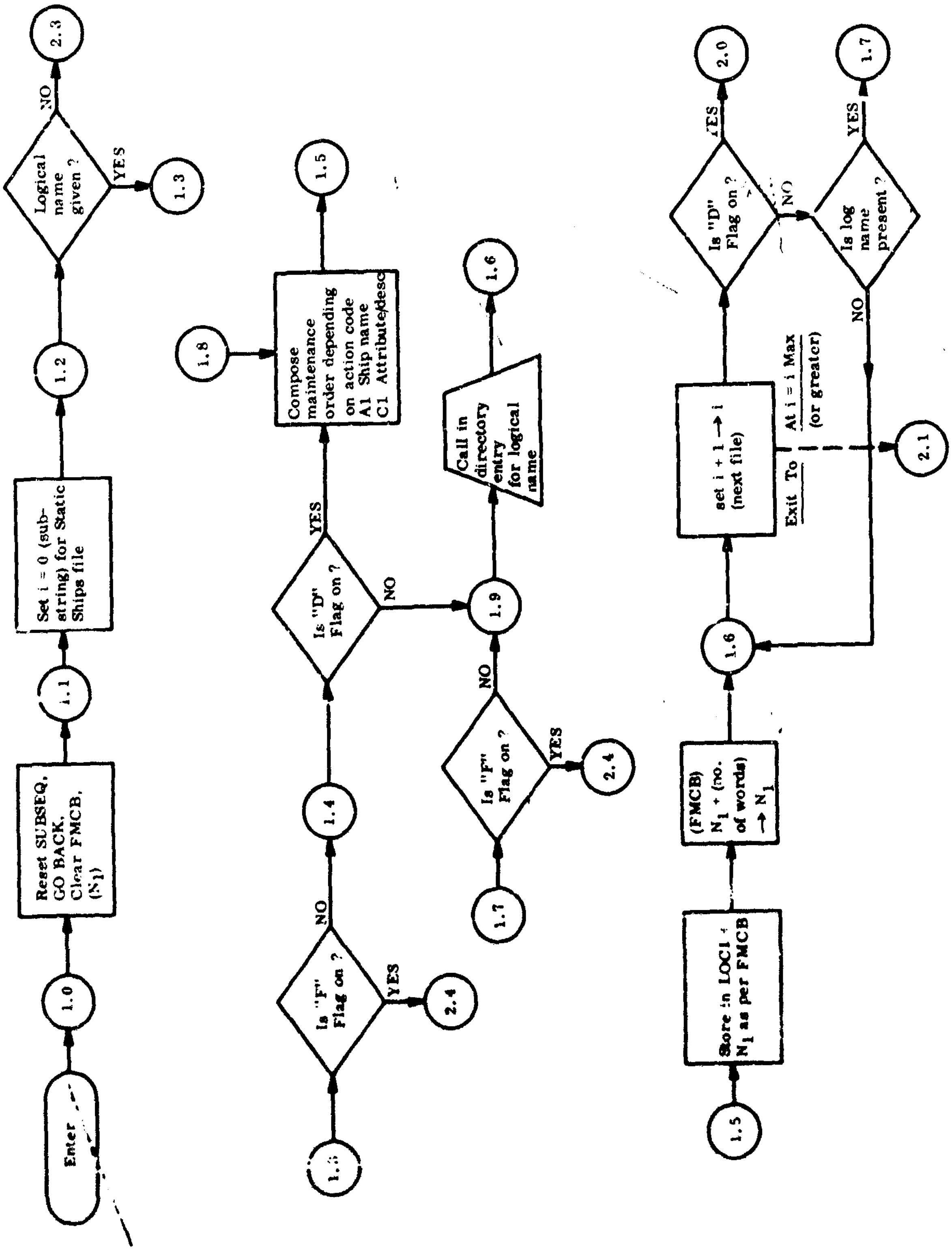

Enter
1.0
Reset SUBSEQ, GO BACK, Clear FMCB, (N1)
1.1
Set i = 0 (sub-string) for Static Ships file
1.2
Logical name given ?
NO
2.3
YES
1.3
1.3
Is "F" Flag on ?
YES
2.4
NO
1.4
Is "D" Flag on ?
YES
Compose maintenance order depending on action code A1 Ship name C1 Attribute/desc
1.8
1.5
NO
1.9
1.7
Is "F" Flag on ?
YES
2.4
NO
Call in directory entry for logical name
1.6
1.5
Store in LOC1 + N1 as per FMCB
(FMCB) N1 + (no. of words) → N1
1.6
set i + 1 → i (next file)
Exit To
At i = i Max (or greater)
2.1
Is "D" Flag on ?
YES
2.0
NO
Is log name present ?
YES
1.7
NO

Figure 5-4. Sub-String Processor (Sheet 1 of 4)

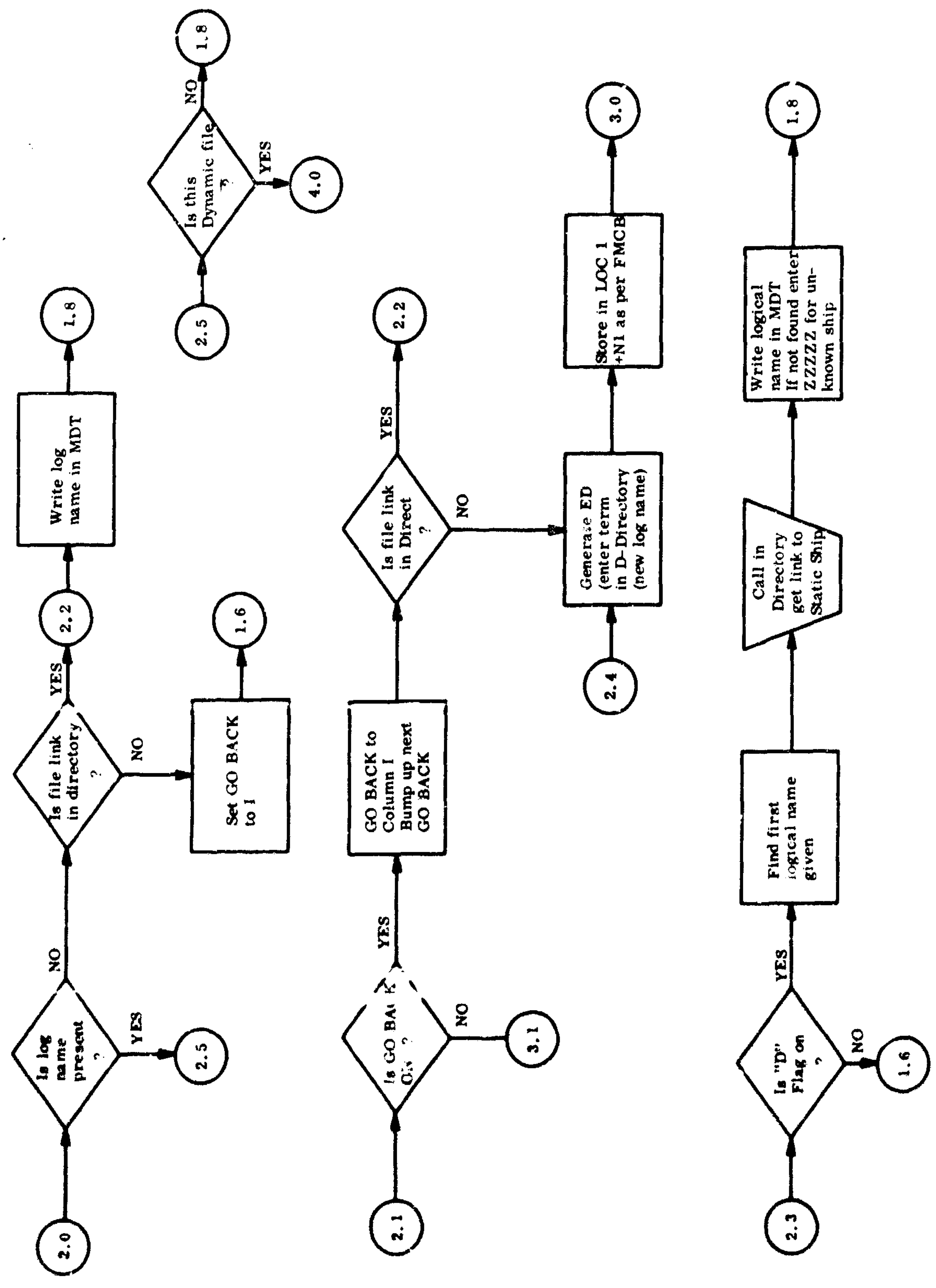

2.0
Is log name present ?
YES
2.5
NO
Is file link in directory ?
YES
2.2
Write log name in MDT
1.8
NO
Set GO BACK to 1
1.6
2.5
Is this Dynamic file ?
YES
4.0
NO
1.8
2.1
Is GO BACK On ?
NO
3.1
YES
GO BACK to Column I Bump up next GO BACK
Is file link in Direct ?
YES
2.2
NO
2.4
Generate ED (enter term in D-Directory (new log name)
Store in LOC 1 +N1 as per FMCB
3.0
2.3
Is "D" Flag on ?
NO
1.6
YES
Find first logical name given
Call in Directory get link to Static Ship
Write logical name in MDT If not found enter ZZZZZ for unknown ship
1.8

Figure 5-4. Sub-String Processor (Sheet 2 of 4)

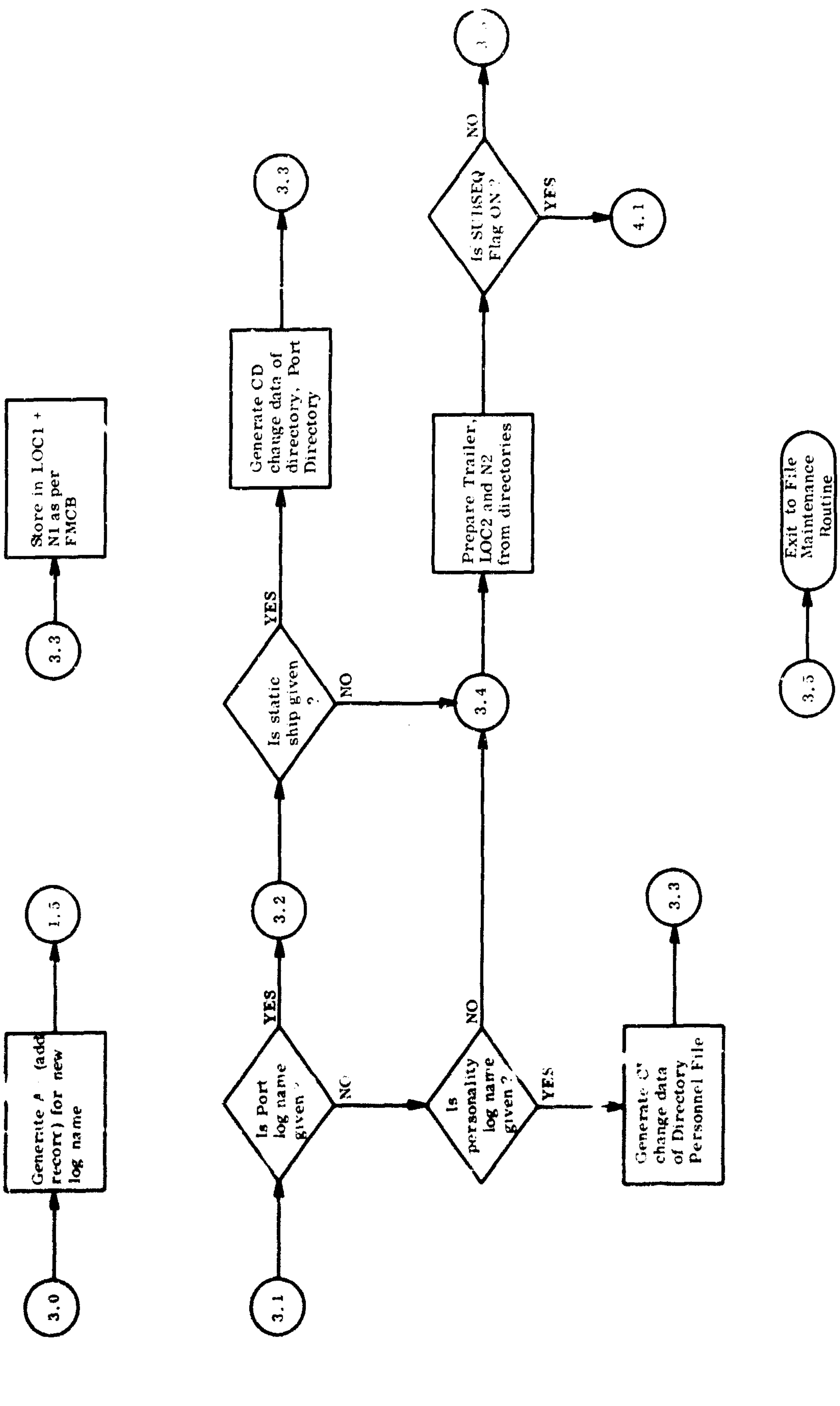

3.0
Generate A (add record) for new log name
1.5
3.3
Store in LOC1 + N1 as per FMCB
3.1
Is Port log name given ?
YES
NO
3.2
Is static ship given ?
YES
NO
Generate CD change data of directory, Port Directory
3.3
Is personality log name given ?
YES
NO
Generate CD change data of Directory Personnel File
3.3
3.4
Prepare Trailer, LOC2 and N2 from directories
Is SUBSEQ Flag ON ?
YES
NO
4.1
3.5
3.5
Exit to File Maintenance Routine

Figure 5-4. Sub-String Processor (Sheet 3 of 4)

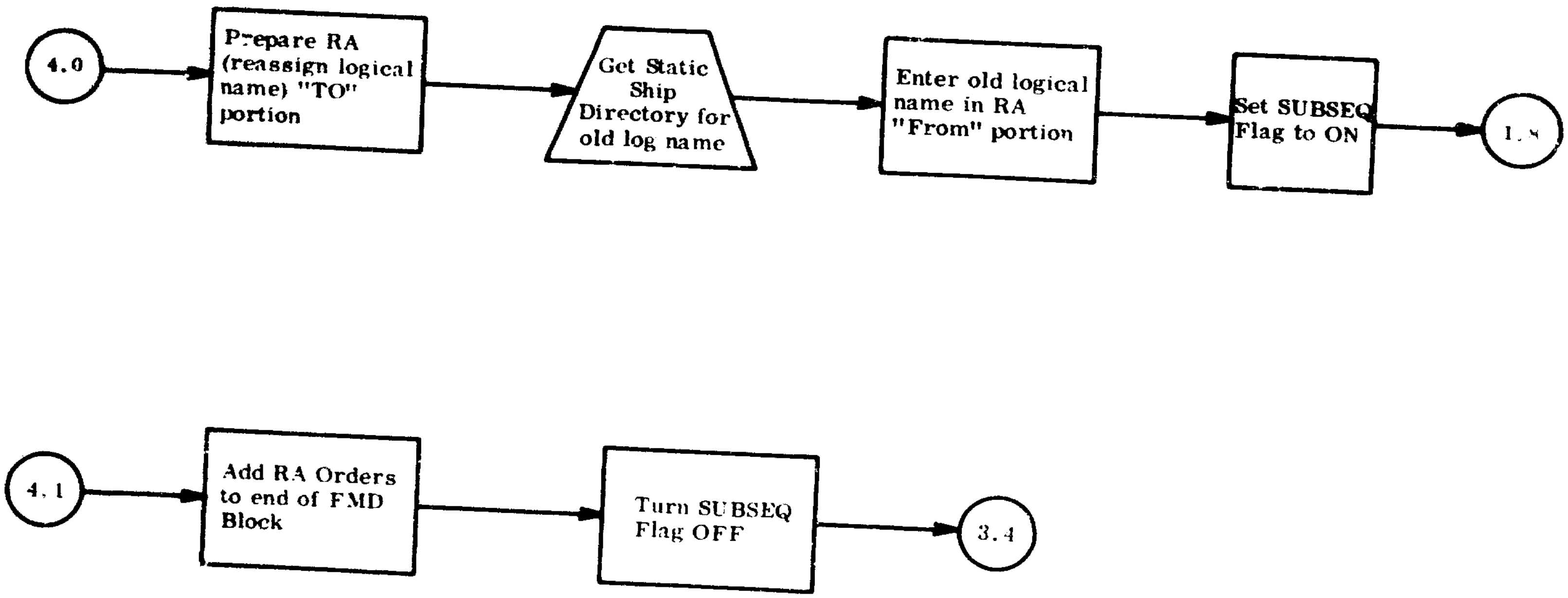

4.0
Prepare RA (reassign logical name) "TO" portion
Get Static Ship Directory for old log name
Enter old logical name in RA "From" portion
Set SUBSEQ Flag to ON
1.8
4.1
Add RA Orders to end of FMD Block
Turn SUBSEQ Flag OFF
3.4

Figure 5-4. Sub-String Processor (Sheet 4 of 4)

5.5 MAINTENANCE ORDER GENERATORS

The Maintenance Order Generator is essentially a program to translate the augmented MDT into file maintenance orders and the file maintenance order trailer(s) required. It is a straightforward program which establishes the linkage with the file maintenance subsystem through:

(1) File Maintenance Orders.

(2) File Maintenance Order Trailers.

(3) File Maintenance Parameter Block.

5.5.1 File Maintenance Orders

The input processor prepares a list of file maintenance orders in the format described in Paragraphs 6.3 to 6.5. This list is located in 1604 core at the time of transfer of control from the input processor to the file maintenance system.

5.5.2 Trailers

In the process of interpreting input messages into file maintenance orders, the input processor performs certain lookups on directories. In order to prevent duplicate lookups by the file maintenance system , pertinent sections of data are held, and cast into the form of maintenance order trailers.

5.5.3 File Maintenance Parameter Block

Because of the unpredictable space requirement for file maintenance orders and trailers resulting from input messages, a parameter block is prepared by the input processor, consisting of the following

Starting at location FMCB
Loc 1:Beginning location of file maintenance orders
N1: Number of words in file maintenance orders
Loc2: Beginning location of trailer information
N2: Number of words in trailer.

5.5.4 Examples

Examples of the file maintenance orders created by the input message processor are given in Figure 5-5.

INPUT MESSAGE	LOOKUPS NECESSARY	RESULTING MAINTENANCE ORDERS
Change: Ship Serial No. XXX, long. N, lat. E, Course y Speed of advance Z	1. D-Directory 2. Static Ship Dir. 3. Dictionary	1. CI (Change data of record), Old Long. Lat. (Logical Name), Course/y, Speed of advance/Z 2. RA (Reassign logical name), Old Long. Lat/ New Long. Lat.
Change: Ship Serial No. XXX, now in Port of Boston	1. D-Directory	1. CD (Change data of a directory), Port Directory, Boston (Logical Name) Ship Serial Number (Logical Name).
Change: Ship Serial No. XXX, to doctor aboard/yes	1. D-Directory 2. Static Ship Dir. 3. Dictionary	1. CI (Change data of record), Operating characteristics record logical name, Doctor on board/yes
Add: Ship Serial AM 3133 name /Valley Forge/ type/CVS/Hull/45/max. speed 33.0/normal speed/ 15.8/no of screws/4/Function/ SUACCAR	1. D-Directory 2. Dictionary	1. ED (Enter term into D-Directory) Ship Serial/AM 3133 Name (Ext) Valley Forge. 2. AR (Static Ships file) AM 3133 (Logical name) type/CVS 3. AR (Prop. Characteristics File) Max. Speed/58/no. of screws/4, Normal Speed/33.0 4. AR (OPIN Characteristic File) Function code/SUACCAR
Add: Ship Serial Unknown, lat. long x, y, speed or advance ZZ, course NN.	1. Dictionary 2. (Note: If Lat. Long. matches, assign logical name from directory)	1. ED (Enter term in D-Directory)/Ship Serial/ U-KKK (special log. name) 2. AR (dynamic File) lat/long (logical name) speed of advance/ZZ; Course NN.

Figure 5-5. Examples of Reduction of Input Messages to Maintenance Orders

SECTION VI. FILE MAINTENANCE

6.1 INTRODUCTION

The data base developed for the sea surveillance system, together with peculiarities of the several hybrid associative systems considered, results in a file maintenance system that appears, initially, to be more complex than is usual. The data base, described in Section IV, is one that frequently requires maintaining more than one data entry as a result of a simple maintenance operation. For example, adding a ship to the file requires, in the worst case, changing seven files, seven file directories, the D-directory, and the available storage file. In reality, however, relatively straightforward means exist for coping with this problem. This section defines the file maintenance problem for the sea surveillance system, presents a general solution to the problem, and develops measures of the utility and efficiency of several hybrid configurations and the 1604 in applying this solution.

File maintenance operations required by the sea surveillance system fall into three classes:

(1) Elementary maintenance operations such as:

- add, delete, or change items in existing records
- add, delete, or change language terms
- update data records with respect to time

(2) Conditional and enumerative applications of maintenance operations such as:

- change an item in all records satisfying a certain Boolean condition
- apply one maintenance rule to all items in a list

(3) File and directory overhaul operations such as:

- create files for the data base
- restructure an existing file

- transfer attributes from one file to another
- reassign logical names

File maintenance operations can be caused by input messages to the system or by file mainten. .ce orders. For convenience, any input message which requires file maintenance is considered as generating a file maintenance order.

For the purpose of this study it is neither necessary nor desirable to explore every file maintenance order in the same level of detail. Some file maintenance operations can be adequately studied in terms of other parts of the system. For example, all file maintenance orders which require conditional or enumerative application of maintenance operations reduces to the application of elementary maintenance operations to outputs from the query subsystem. Since means are at hand for evaluating the query subsystem, evaluation of maintenance operations of this class can take place once the elementary maintenance operations have been evaluated. Hence, these operations are not explored to the level of detail that would be required to actually implement the system.

Many of the file maintenance operations which are classed as file and directory overhaul operations need not be explored in great detail for another reason. Although important components of a completely general sea surveillance system, only one of the tasks mentioned has day-to-day operational significance: reassigning logical names. The tasks of file creation and file restructuring are "once-in awhile" tasks, and evaluating them in detail can therefore be regarded as a very low-priority item for a study of this kind.

File maintenance in an operational sea surveillance system can, and probably does, require considerable data control. In any actual environment, exhaustive logic checks of file maintenance orders, checks for reasonableness of reports (e.g., can a ship possibly reach a newly reported location in the time elapsed from the last report?), and carefully constructed "histories" to make possible rapid recovery from an erroneous change can be expected. Such data controls are not employed in this study for two reasons: all the controls are not known that must be applied; second, such controls are bound to be formally very much like other operations which are required by the input message processing and query subsystems. For the purpose of this study, it is assumed that file maintenance orders are formal, temporary, and, with respect to content, correct.

The elementary operation "update with respect to time" consists of the performance of a routine, or series of routines, which would, among other things, apply dead reckoning. Obviously, dead reckoning would be applied to only those records for which more reliable data is not available. The application of such routine(s) can be studied by analogy with the various function calculating routines discussed elsewhere; a detailed study of this function of file maintenance need not, therefore, be performed.

The preceding considerations justify limiting the detailed investigation of file maintenance operations to those elementary operations which add, delete, or change items in existing records; add new records; add, delete, or change language terms; and reassign logical names. Once each of these file maintenance operations has been explored in detail, inferences, based on results of the query and input message processing subsystems, will be employed to evaluate, in general, file maintenance operations. Evaluation of file creation and file restructuring operations is not made because of the low priority and expected low return of such a study.

6.2 FILE MAINTENANCE PROBLEMS PECULIAR TO THIS SYSTEM

An input message which states that a ship has entered port will generate a maintenance order affecting the port file directory. In the present system, however, it is important to note that there are often other files that must also be maintained as a result of the order. For example, when the ship is in port, the dynamic ship file must not assign a speed and direction to the ship since it is at rest, and the system must assure that the latitude and longitude of the ship correspond to that of the port. It should also be noted that the static ship directory must reflect a reference to the appropriate port entry.

It must be recognized that the directories in this system are not only means to obtain data quickly, but they contain data, and must be treated as files (or sub-files). Hence, an input message calling for an add, delete, or change of a data record (or an item within the record) will very often generate an add, delete, or change for the corresponding directory. Because the static ship directory provides links for all files having data about a particular ship, a change of the static ship directory must often be generated.

6.3 DESCRIPTION OF THE FILE MAINTENANCE ORDER

The file maintenance order always contains the following components:

(1) A transaction type identifier indicating the kind of maintenance operations required (e.g., add a record, delete an item).

(2) A record selector component which, for the purpose of detailed study, can be regarded as a name, although in the general case records can also be selected by enumerator clauses and Boolean conditions. Note that the selector component must also be able to select a directory entry.

(3) An operand component consisting of a list — usually a list of pairs — of information to be operated upon. For example, when the transaction type is an "add data to record," the operand component is the attribute-descriptor pair(s) to be added.

In the course of interpreting an input message and preparing a file maintenance order, the input message processing subsystem frequently must consult directories, dictionaries, and even data records. Each of these operations requires a disc access by the input message processor. To minimize the duplication of these accesses, the input message subsystem saves referenced data in what is called the trailer portion of the file maintenance order.

The trailer portion of the file maintenance order is in the following form:

(1) The first word of each subrecord identifies the file or directory from which the data comes, and contains a "tag field" indicating that the word begins a section of stored data.

(2) The second word contains the physical location from which the data was obtained.

(3) The third and subsequent words contain the data.

The trailer portion of the file maintenance order consists of a series of records, each having the above format.

For convenience of file maintenance, the smallest amount of data saved is always at least one disc block in length, and is an integral multiple of disc block length.

6.4 FILE MAINTENANCE ORDER FAMILIES

A single output message to the sea surveillance system may require action on several files. One reason for this has already been noted; a second reason is that an input message reporting on a ship can contain several attribute-descriptor pairs, where the attributes are to be found in different files. To speed up processing by file maintenance, all file maintenance orders recognized by input processing as applying to one ship are passed on as members of one family.

The family (which in the degenerate case can have only one member) is the unit processed by the file maintenance system.

6.5 TRANSACTION CODES

This paragraph lists those transaction codes which were studied in detail in the evaluation. The list is known to be incomplete, for reasons stated in Paragraph 6.1. The selection of codes and their mnemonics is arbitrary.

Code	Meaning
AI	Add data to record
AD	Add data to directory
DI	Delete data from a record
DD	Delete data from a directory
CI	Change data of a record
CD	Change data of a directory
AR	Add a record
DR	Delete a record
CR	Change a record (replace entire record)
RA	Reassign a logical name
ED	Enter a term into the D-directory
DX	Delete a term from the D-directory

Transaction codes fall into three groups:

Group I	AI; DI; CI; AD; DD; CD
Group II	AR; DR; CR
Group III	RA; ED; DX

File maintenance processing varies significantly by group, and less significantly within group.

6.6 FILE MAINTENANCE PROCESSING

File maintenance processing for this system involves four basic steps:

(1) Assemble all records and directory entries that can be affected by file maintenance order.

(2) Execute the basic file maintenance operation called for.

(3) Execute any file maintenance operations on other files or directories required by the file maintenance order.

(4) Iterate through all file maintenance orders within the family and write out the results when the entire family has been processed.

Depending on whether an associative memory is available, there are slight but significant variations in the manner in which these steps are performed.

The following basic routines are required:

(1) The Assembler which gathers together all the record and directory entries required for executing the file maintenance order into the trailer, based on the indicator bits contained in the dictionary entry for the appropriate attribute(s); constructs an action table, which is a table of relative locations of the entries in the trailer; and performs certain minor housekeeping functions.

(2) The Router which interprets the file maintenance order and calls the appropriate executing routine.

(3) The Executing Routines of which there are three, one for each group of transaction codes, ER1, ER2, and ER3:

(a) ER1 performs the functions appropriate to transaction codes of Group I: add,data, delete data, change data; add , delete , or change entries in the directories (except for the D-directory) .

(b) ER2 performs the functions appropriate to transaction codes of Group II: add, delete, or change (substitute) records.

(c) ER3 performs the functions appropriate to transaction codes of Group III: reassign logical names, enter terms in the D-Directory, delete terms from the D-Directory.

(4) The Related Transaction Routine which examines the file maintenance order and, using a table of possible related file maintenance requirements, determines what other file maintenance may be required as a result of the file maintenance order. It then tests conditions and, if conditions hold that require an additional maintenance order, it generates a file maintenance order and calls the Router. The Router in turn calls the appropriate routine to execute the order, after which control is returned to the Related Transaction Routine.

(5) The Family Control Routine which serves the function of assuring that all file maintenance orders of the same family have been executed. If not, it obtains the next file maintenance order for the family and transfers control to the Assembler. If all orders of the same family have been executed, the Family Control Routine exits to the Executive.

The Executive is not discussed in this section. It is assumed that, when a file maintenance order family requires attention, the Executive transfers control to the file maintenance subsystem. When the family has been processed, control is returned to the Executive.

Processing flow is presented in Figure 6-1.

6.7 ASSEMBLER

Input to the Assembler consists of the file maintenance order (FMO) and its trailer. From this input, the Assembler creates an action table and inserts into the trailer any other entries that may be required to complete the FMO.

The action table is a list of entries defining the data contained in the complete trailer, its relative location in memory, and whether the data in question has been

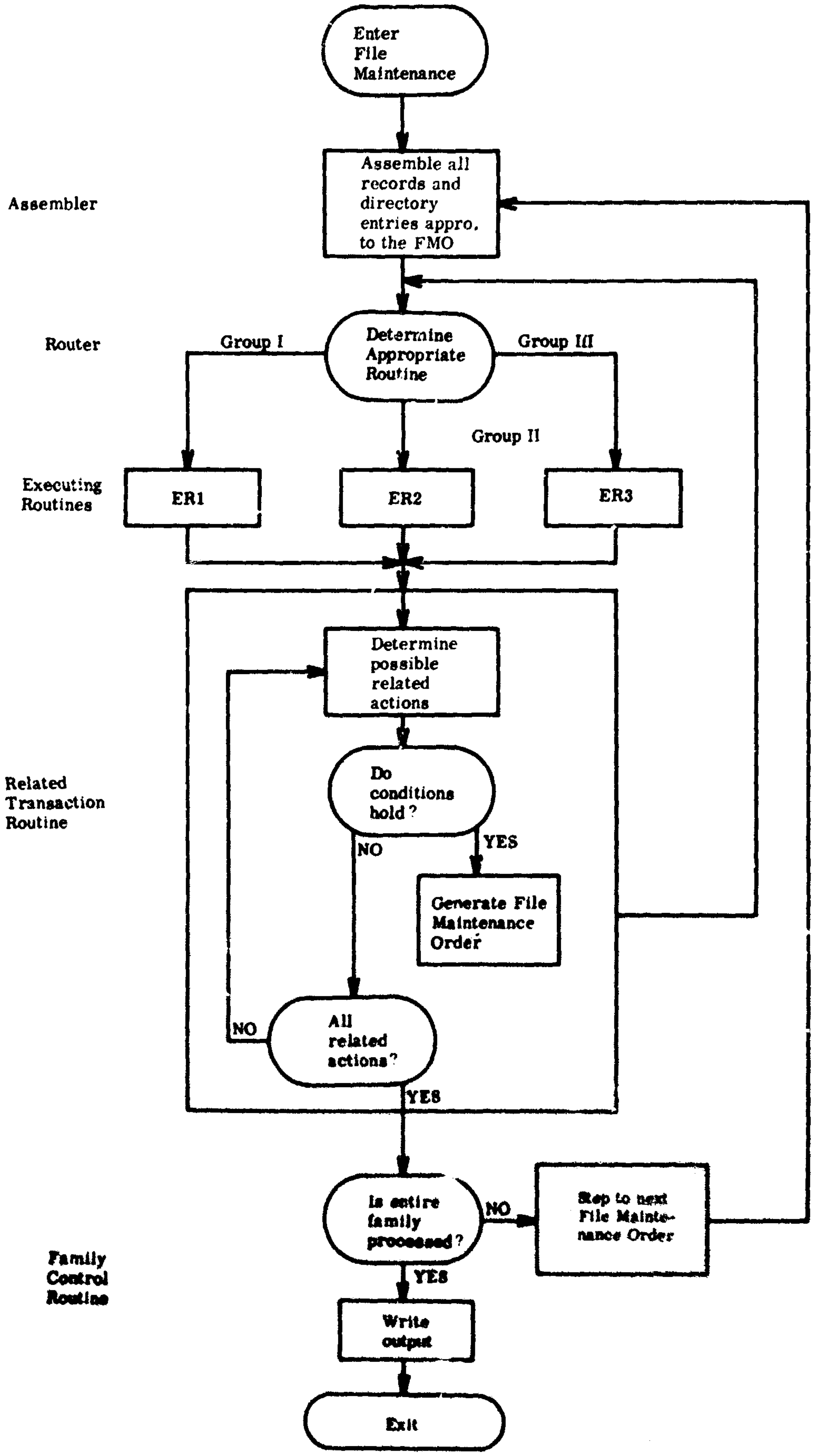

Enter File Maintenance
Assembler
Assemble all records and directory entries appro. to the FMO
Router
Group I
Determine Appropriate Routine
Group III
Group II
Executing Routines
ER1
ER2
ER3
Determine possible related actions
Related Transaction Routine
Do conditions hold?
YES
NO
Generate File Maintenance Order
NO
All related actions?
YES
Is entire family processed?
NO
Step to next File Maintenance Order
Family Control Routine
YES
Write output
Exit

Figure 6-1. File Maintenance Processing Flow (General)

changed by the file maintenance subsystem. It is composed of entries of the following form

A D F F L L L L

where each letter identifies a character position, and

A = 1 if the entry in question has been changed by the file maintenance subsystem

0 if the entry in question has not been changed.

D = 1 if the entry in question is a file directory entry

2 if the entry in question is a D- directory entry

3 if the entry in question is a dictionary entry

0 if the entry in question is a data file record.

F = the file code to identify the data file from which the entry was obtained, or, if the entry is a file directory entry, which file it is a directory entry for.

L = the absolute machine address of the entry if the entry is in the 1604, and 77777777_8 if the entry is in the associative memory (GAP).

The variable "A" superficially defines whether the corresponding entry has been changed by the file maintenance subsystem. Its primary purpose, however, is to minimize disc writes. The file maintenance subsystem will not write out any data unless the data has actually been changed by processing. In writing out data, the action table is consulted and the records are written only if A = 1; hence, no more records are written than are actually required. The variables D, F, and L are self explanatory.

One important use of the action table deserves notice. The file maintenance subsystem processes FMOs within families. The action table is built up, therefore, for all entries required for the entire family. The routine that creates it first consults the existing action table, if any, in order to determine whether an entry needed for one FMO of a family is already available, owing to the results of processing some prior FMO of the same family. If such an entry has been made, the record or file directory entry does not need to be read in from disc. Hence, the action table also minimizes disc reads. In the case of the dictionary and/or D-directory entries required by an

FMO, the case is somewhat more complex. Simply because a dictionary or D-directory entry has been made, there is no assurance that the corresponding entry is the one required for this particular FMO. After determining that a dictionary or D-directory entry has been made, the Assembler must determine if the corresponding trailer entry is satisfactory for processing the FMO in question. In such a system, of course, there can be many D-directory and dictionary entries for a family, and the Assembler must step through all such entries before concluding that a disc read is required.

The FMO contains the transaction type identifier, a record selector component, and an operand component (see Paragraph 6.3.1). The transaction type identifier is the first word in the FMO, and the FMO is the first block of data received by the file maintenance subsystem.

The transaction type identifier contains two hexabit coded characters in the low-ordered position (bits 0 through 11) of a word otherwise filled with zeros. These two characters are selected from the codes listed in Paragraph 6.5.1, and are interpreted by the system as transaction codes.

The second word in the FMO contains the record selector component. Depending upon the transaction code, the record selector component is given a different interpretation. The rules for interpreting the record selector component are given in Table 6-1.

TABLE 6-1. RULES FOR INTERPRETING RECORD SELECTOR

If the transaction code is	Interpret the record selector as
AI DI CI	Logical name of a record
AR DR CR	Logical name of a record
AD DD CD	File directory code or attribute identifier
RA	Logical name of a record
ED DX	Character count of D-directory entry

A file directory code is an eight-bit code in the standard file identification convention appearing in the low-order position of a word. The remainder of the word is zeros; hence, a file directory code can be distinguished from an attribute identifier. A character count of a D-directory entry is simply the count, in binary, of the number of characters in the subsequent D-directory entry.

The third and last component of the FMO is the operand component. The interpretation of the operand component varies according to the transaction code and record selector.

If the transaction code is AI or DI, the operand component is always one attribute-descriptor pair. In this event, the operand component contains the attribute-descriptor pair to be added to or deleted from the record identified by the record selector component. If the transaction code is CI, the operand component consists of two attribute-descriptor pairs; the first identifies the attribute-descriptor pair before the change* and the second identifies the attribute-descriptor pair after the change.

* the event that this is not known, the field will be filled with blanks.

If the transaction code is AR, DR, or CR, the operand component consists of an entire data record.

If the transaction code is AD or DD and the record selector component is a file directory code, the operand component of the FMO is an attribute name. The FMO is interpreted to mean that this attribute name is to be added to, or deleted from, the stated directory. If the transaction code is CD and the record selector component is a file directory, two attribute names are given, and the command is interpreted to mean that the first attribute name is to be changed to the second attribute name.

If the transaction code is AD, DD, or CD and the record selector component is an attribute identifier, the FMO orders that a revision of the dictionary be made. Such an FMO is generated internally by the system in light of an FMO of transaction code AD, DD, CD, RA, ED, or DX. In each case, the operand component is either a dictionary entry, a pair of dictionary entries, or an action table entry followed by a dictionary entry. Rules governing this situation will become clear after processing has been defined further.

If the transaction code is ED, DX, or RA, the operand component is a logical name in external form, followed by one word of zeros, followed by a logical name in internal format.

The trailer to the FMO consists of a list of entries of the following kind:

- Data File Records
- Dictionary Entries
- Data File Directory Entries
- D-Directory Entries

The entries consist of separate disc records (in their entirety) preceded by a disc address. The disc address is, in turn, preceded by a flag word of all "ones" (minus zero). It follows that entries, together with their addresses and flags, occupy 34-word modules.

The logic of the Assembler is presented in Figure 6-2. Briefly, the Assembler involves five distinct steps.

(1) Determine whether the current FMO is the first FMO of a new family. If it is, clear the old action table, and set an indicator variable, FAMIND, to one. FAMIND is set to zero when the family control routine determines that an entire family has been processed.

(2) An initial action table is created. Bring all FMO trailers into one trailer, and stack the FMOs into one list.

(3) Determine, from the relevant file indicators, whether all data records and data file directory entries have been read into the trailer. This test consists of forming the logical "or" of all relevant file indicators, and determining from the action table that both a file and file directory entry are present for each bit whose value is one.

(4) Assure that every attribute in the FMO has a corresponding dictionary entry in the trailer. In this test, it is necessary to use both the action table and the actual contents of the trailer. In the event that no such entry exists in the trailer, the dictionary is read until the appropriate entry is found, and then this entry is stored in the trailer. A corresponding action table entry is created.

(5) Step three is repeated for D-directory entries for all logical names.

6.8 ROUTER

The Router examines the transaction codes contained in the transaction type identifier and calls either ER1, ER2, or ER3, as appropriate. The processing of the Router -- essentially a series of tests — is presented in Figure 6-3.

6.9 EXECUTING ROUTINES

There are three Executing Routines, ER1, ER2, and ER3. Each routine handles a group of transactions based upon the transaction code of the current FMO.

Each Executing Routine has the same general format. The routines begin by performing a series of tests upon the transaction code and upon the record selector portions of the FMO. Based upon these tests, a subroutine of ER1, ER2, or ER3 is

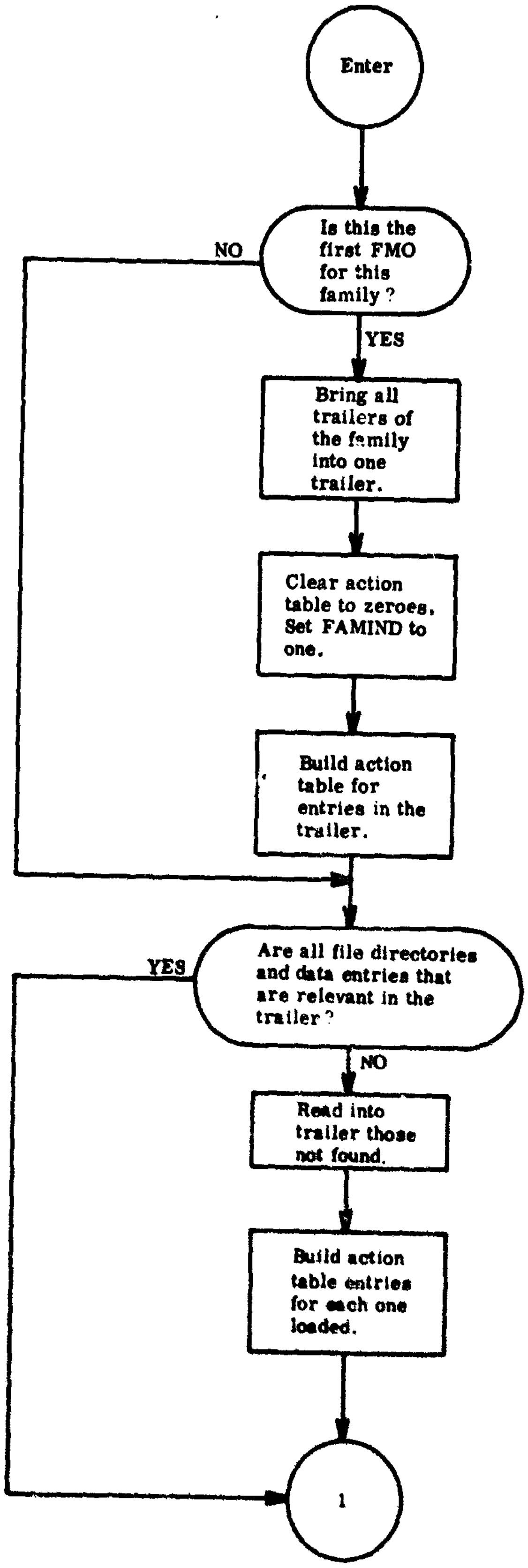

Enter
Is this the first FMO for this family ?
NO
YES
Bring all trailers of the family into one trailer.
Clear action table to zeroes. Set FAMIND to one.
Build action table for entries in the trailer.
Are all file directories and data entries that are relevant in the trailer ?
YES
NO
Read into trailer those not found.
Build action table entries for each one loaded.
1

Figure 6-2. Assembler (Sheet 1 of 2)

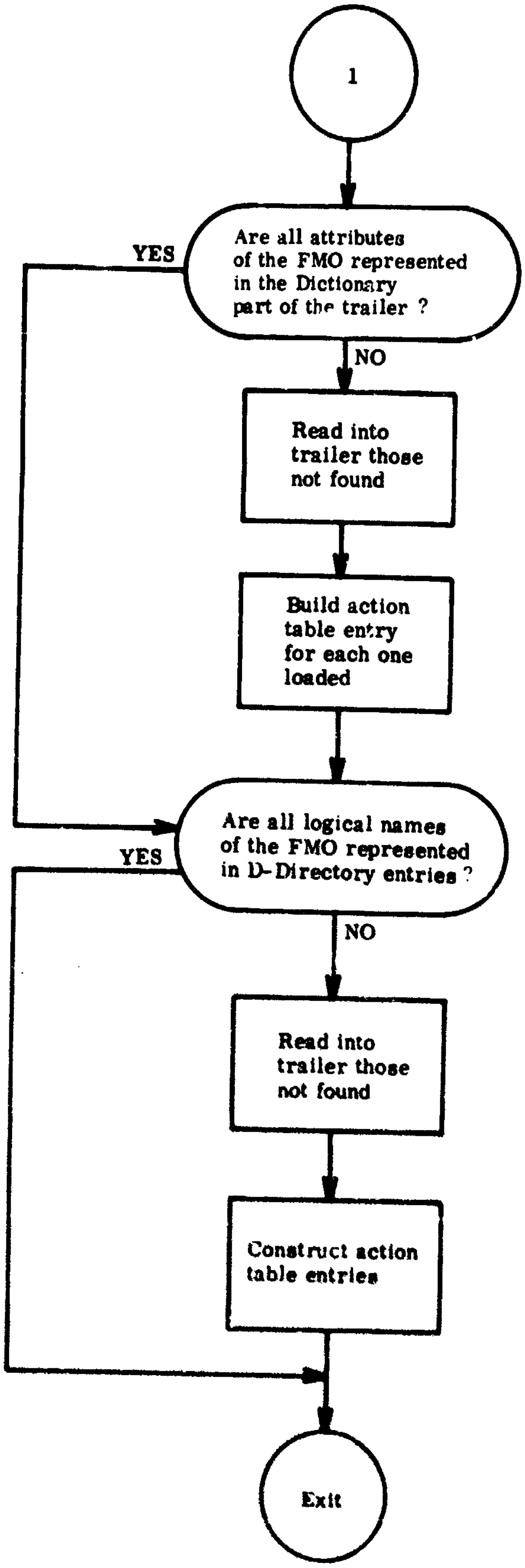

1
Are all attributes of the FMO represented in the Dictionary part of the trailer ?
YES
NO
Read into trailer those not found
Build action table entry for each one loaded
Are all logical names of the FMO represented in D-Directory entries ?
YES
NO
Read into trailer those not found
Construct action table entries
Exit

Figure 6-2. Assembler (Sheet 2 of 2)

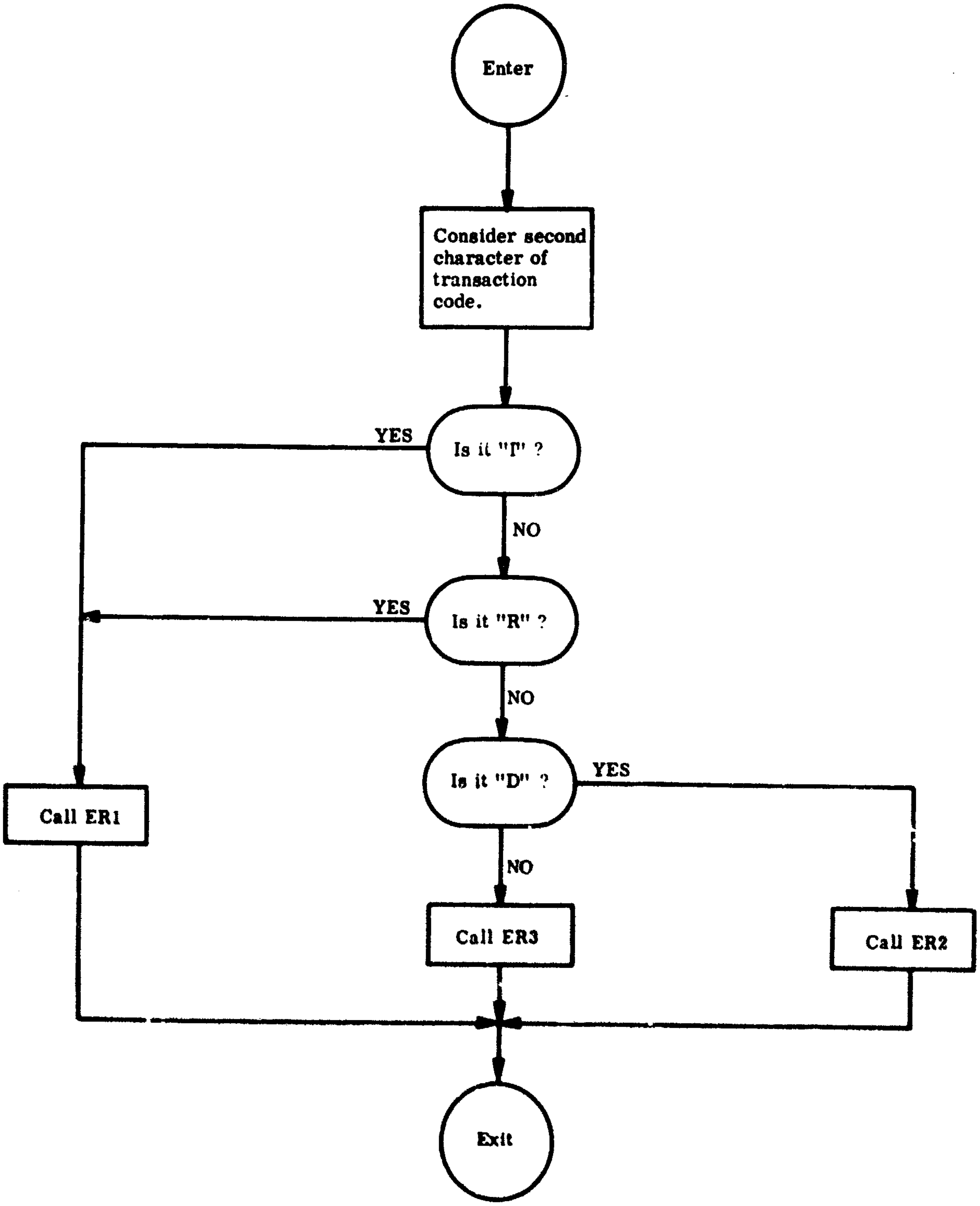


Figure 6-3. Router

called upon. These subroutines correspond to transaction codes or, in several cases, transactions code-selector component pairs. Some of the subroutines, such as the change item routine, in turn, consist of calls to other routines. The change item routine consists, essentially, of a call on the delete item routine, followed by a call to the add item routine. The overall logic of ER1, ER2, and ER3 is presented in Figures 6-4, 6-5, and 6-6, respectively.

There are significant differences in the manner in which Executing Routines handle a task, depending on whether or not an associative memory is available. These differences are studied in this paragraph.

6.9.1 ER1

ER1 consists of an ER1 router and the following subroutines:

(1) Add data (ERAI)

(2) Delete data (ERDI)

(3) Change data (ERCI)

(4) Add record (ERAR)

(5) Delete record (ERDR)

(6) Change record (ERCR)

6.9.1.1 ERAI. A selected perform operation (SPO) is a macro operation utilized in input message processing and file maintenance. Input parameters to SPO specify the operation to be performed, whether to perform the operation on responders or on data between two addresses, and, if between addresses, what the beginning and ending addresses are. Briefly, SPO permits the programmer to treat some subset of the AM as if it were the entire AM. SPO can be utilized to produce a version of ERDI that is almost competitive with that described in Paragraph 6.9.1.2. SPO is vital to ER1, ER2, and ER3 in other operations. As employed in the remainder of this discussion, SPO can be regarded as subsetting the AM.

In adding data using the 1604-B, the action table is consulted to find the first available space in either the record or the last continuation record. If no space is available, a continuation record is written and the data inserted. This operation is

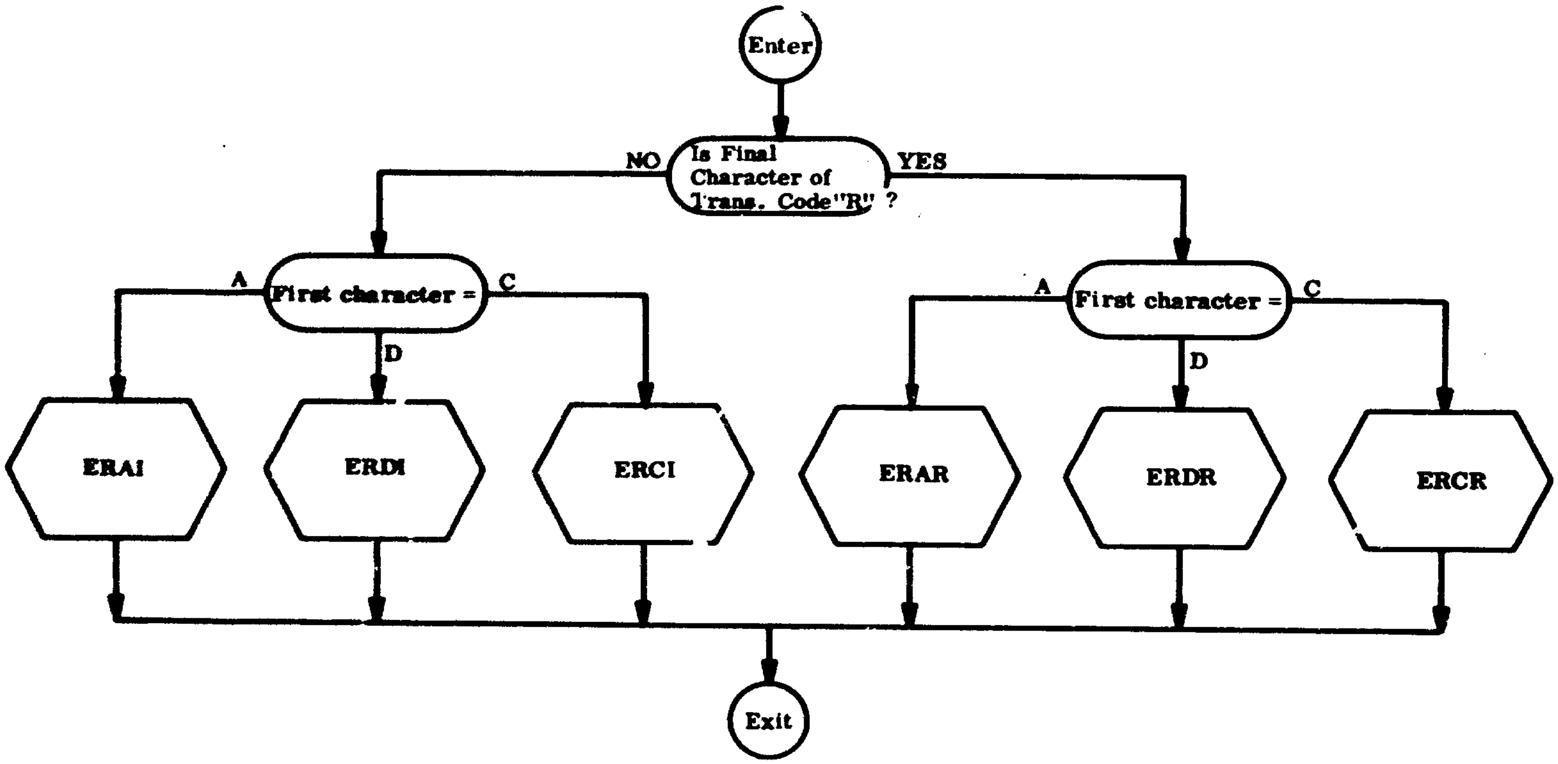


Figure 6-4. ER1 Routine

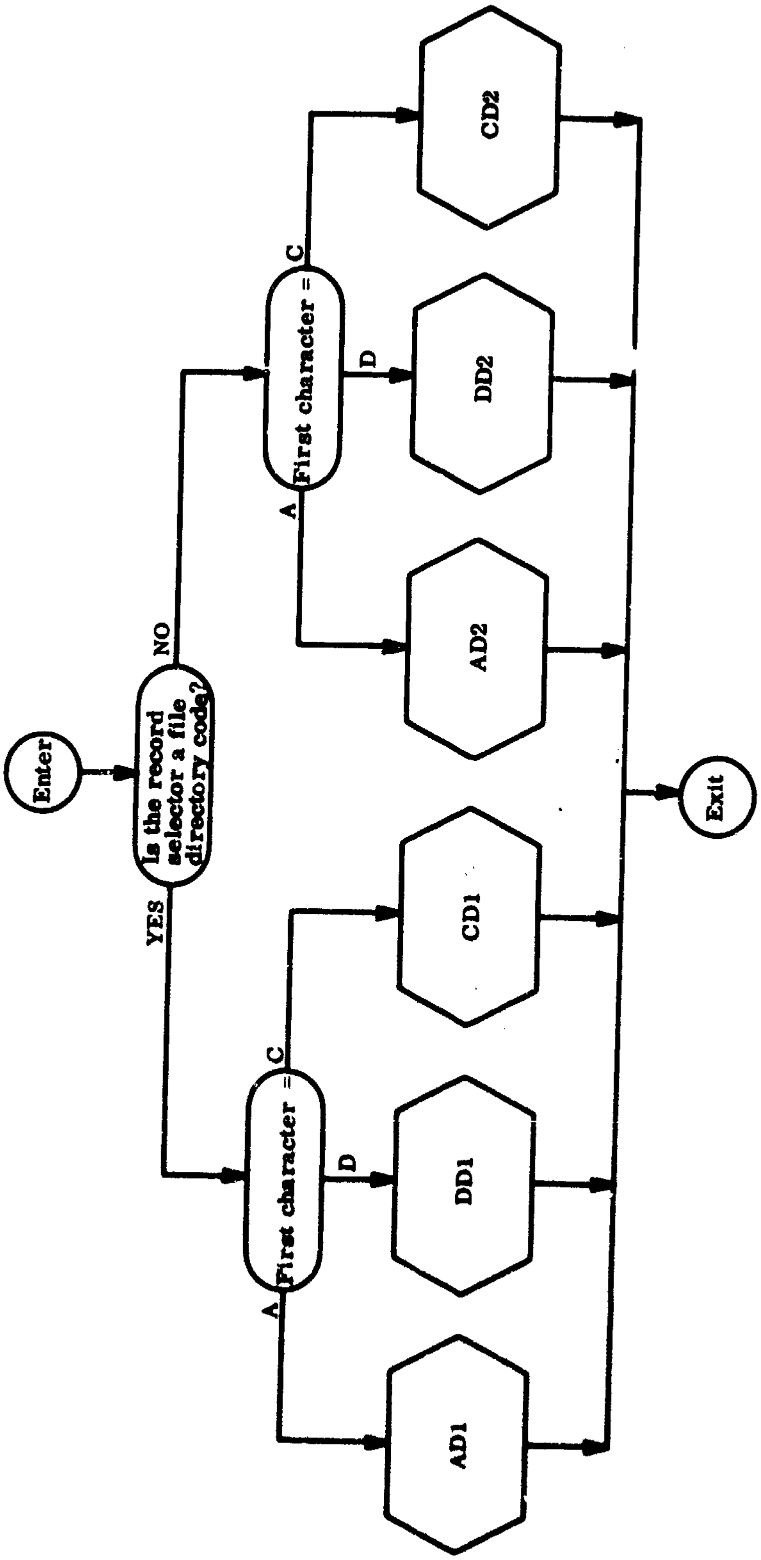


Figure 6-5. ER2 Routine

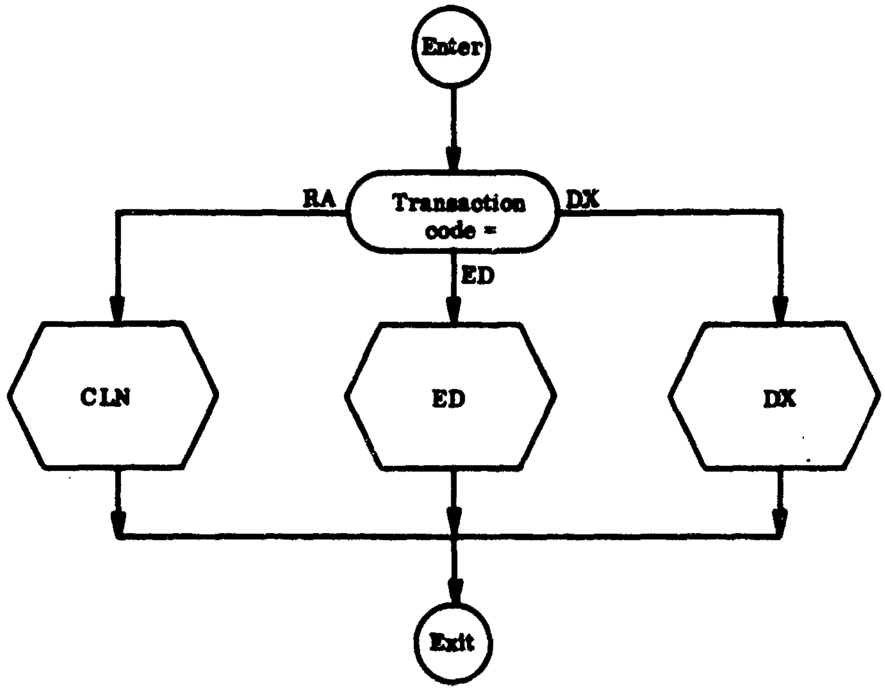


Figure 6-6. ER3 Routine

harder to program than one might believe at first glance. Note that a directory entry for the continuation record, and a D-directory entry, must be created.

In the AM, SPO is used to subset the AM to the record and its continuation record. A search for n consecutive words of zeros, where n is the number of words in the data to be added, is performed. After the search is completed, the program exits from SPO. If a space is found available, data is then written into that location. If no available space is found, the flag word and logical name followed by the data are written, again using WFA, when the first available space in the AM is found.

Again, the dictionary and D-directory must be updated.

Where the 1604-B programmer must step down the records to find available space, the AM programmer can find it, if it exists, in essentially one step.* If it does not exist, the WFA instruction greatly simplifies his task of writing a new continuation record. Hence, he has an easier programming task. Further, while it is extremely difficult to give a time estimate for ERAI, because timing will vary with the number of words in the data, the number of words in the trailer, the distribution and number of words in the affected record, and the relative locations of the affected records, it is still reasonable to say that the GAP version of ERAI is sufficiently good from the efficiency point of view to be preferred to the 1604-B only version.

The utility of SPO in this operation should not go unrecognized. Because adding and changing data are large parts of any intelligence file maintenance operation, and because ERAI is essential to adding and changing (see Paragraph 6.9.1.1), there is some reason to believe that it will be very frequently used in any such operational system. SPO was actually developed to overcome certain limitations in GAP for input message processing. It is recommended that SPO be considered for inclusion in an AM, as hardware, or, if this is not possible, instructions should be included to make SPO easy for the programmer to use. Note that the A2 memory would obviate the need for SPO.

* This assumes a SPO that permits n consecutive search-shifts, which is easily obtained. Otherwise, n steps are required.

6.9.1.2 ERDI. Deleting data from a record is a more complex programming task in a conventional memory than in an associative configuration such as GAP. Surprisingly, it is less efficient when done in the GAP. This will become evident after the tasks that must be done in ERDI for the 1604-B have been stated (see Figure 6-7).

When ERDI is executed in the 1604-B, the action table is used to find the appropriate record. A search is then performed to find the beginning and ending memory locations of the attribute-descriptor pair. All remaining parts of the record are then moved up in memory to write over the attribute-descriptor pair to be deleted. If the record has a continuation record, data from the continuation record is then moved up (if possible). In the event that such a move from a continuation record to its predecessor results in a continuation record lacking data (i.e., containing only a logical name), ERDR is called in to assure that the corresponding record is deleted from the data base.

In the Goodyear Associative Processor the programming is simpler. The attribute-descriptor pair to be deleted is located and erased. The continuation record(s), if any, is located. Its flag word and logical name are erased, and the original record and continuation record(s) are read out of the AM into the 1604-B, using the RUM instruction. The records in toto, are erased from the AM, and the 1604-B records are fed into the AM in 32 word units, preceded by the flag word and logical name, using WIA on non-zero words in the 1604-B.

Although the AM procedure above is, at least from one point of view, conceptually simpler in handling the problems created by continuation records, it is more time consuming than a 1604-B algorithm. Because A3 does not require that data be passed to and from disc through 1604-B memory, use of the A3 rather than GAP will result in a more efficient algorithm competitive with the 1604-B. With reservations to be explained in the following paragraph, the following results can be stated.

(1) From the point of view of certain "housekeeping" problems, ERDI is easier to program in an associative processor.

(2) A GAP version of ERDI will be more time consuming, and hence less efficient, than a conventional memory version.

(3) An A3 version of ERDI will be less time consuming than either GAP or a conventional memory version since transfer times from disc to AM and AM to disc are eliminated.

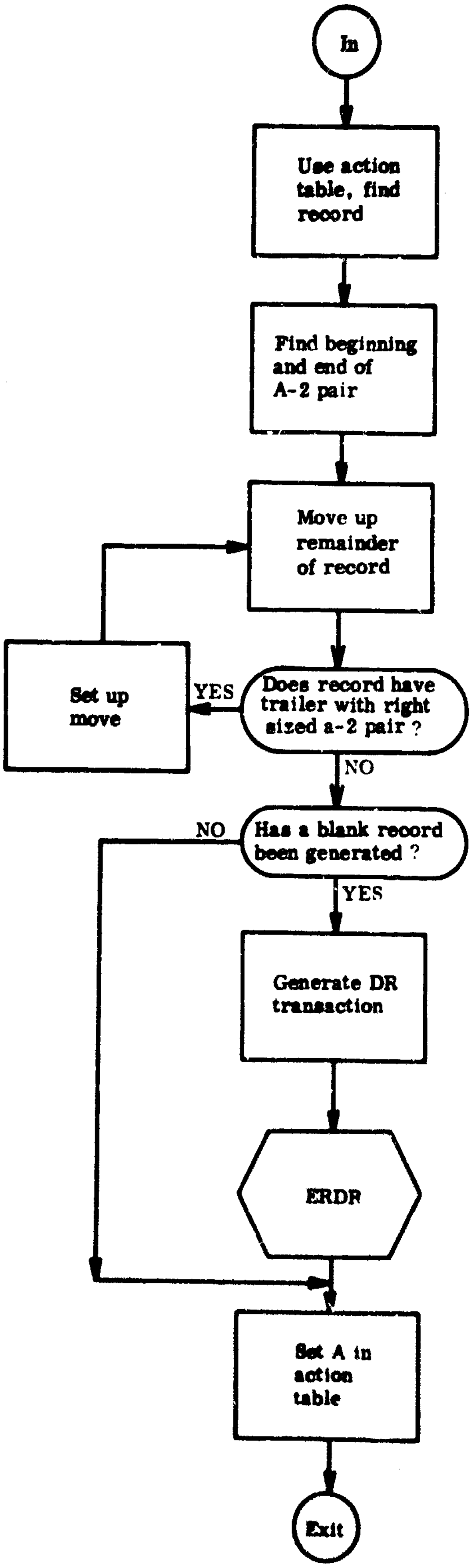

In
Use action table, find record
Find beginning and end of A-2 pair
Move up remainder of record
Does record have trailer with right sized a-2 pair ?
YES
Set up move
NO
Has a blank record been generated ?
NO
YES
Generate DR transaction
ERDR
Set A in action table
Exit

Figure 6-7. ERD1 Routine

The following reservations must be kept in mind in reviewing these conclusions:

(1) It is possible to design a system in which all delete data transactions are performed at one time (i.e., a list of DI transactions is to be executed, rather than one DI at a time). In such a system, the GAP would approach the 1604-B is efficiency.

(2) With certain added instructions, or modifications to existing instructions, GAP would be even easier to program for ERDI, and would be more efficient. For example, if RUM had an option to read out only live data, rather than read out zeroes for non-live data, the routine would be easier to write and more efficient. Adding an instruction with such power would be equally good.

6.9 1.3 ERCI. Figure 6-8 shows that to change data one performs ERDI followed by ERAI; hence, this routine is not discussed further. It should be understood, however, that the discussion on ERAI and ERDI also applies to ERCI.

6.9.1.4 ERAR. The addition of a record is far simpler and far less time consuming in the GAP and A3 than in the 1604-B only configuration. The reason for this is the availability of the WFA instruction which finds the first available space in the trailer and writes the record. The housekeeping steps provided for continuation records are the same, or similar, as are steps involved in the D-directory and file directory maintenance operations. Hence, by virtue of the WFA instruction alone, the addition of a record is best accomplished in the associative memory.

6.9.1.5 ERDR. Deletion of a record in the AM, as in the 1604-B requires that zeroes be written in the disc record positions. In the AM, an erase instruction is used, and, at the end of the family control routine, the AM records, under control of the action table, are read out using RUM. In A3, these records can, essentially, be written directly to disc. In GAP, these records must be read into the 1604-B, and then into disc. Because of the load-unload times of GAP, it is therefore notably less efficient than either A3 or the 1604-B.

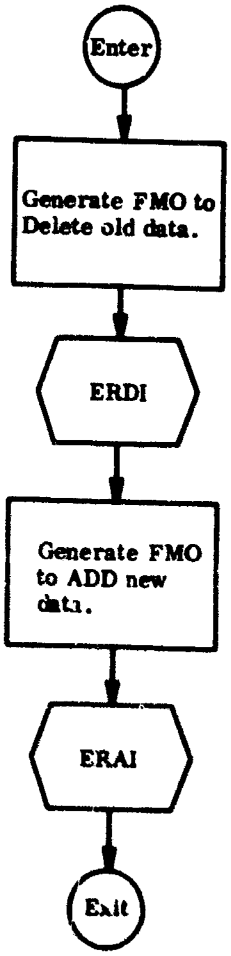

Enter
Generate FMO to Delete old data.
ERDI
Generate FMO to ADD new data.
ERAI
Exit

Figure 6-8. ERC1

6.9.1.6 ERCR. A change (substitute) record operation is simply a delete record followed by an add record. It need only be noted, therefore, that the discussion in Paragraphs 6.9.1.4 and 6.9.1.5 apply.

6.9.2 ER2

ER2 adds directory entries or changes or deletes existing entries. Many of the FMOs executed by ER2 are generated by routines within ER1. For example, the deletion of a record or the addition of a record causes the deletion or addition of the corresponding directory entry.

Depending on whether the record selector component is a file directory code or an attribute identifier, the addition, deletion, or change is performed on either the file directory entry or the dictionary. For the purpose of understanding the relative value of the AM, only those operations that affect the file directory need to be considered here. The reason for this is as follows: While the deletion of an attribute or the changing of an attribute in the dictionary involves a dictionary maintenance operation, it also generates FMOs affecting every occurrence of the attribute in a file directory or in a data record. Hence, the actual maintenance of the dictionary is the least important part of carrying out such an operation, and constitutes perhaps 1/100th of the time required to carry out an operation of this kind in a full sea surveillance system.*

One important comment must be made about these routines. In order to process deletions or changes to a dictionary entry, it is necessary to provide for the possibility that the trailer will exceed the capacity of the AM. For this reason, these routines must run under control of the AM dispatcher, just as portions of the query system must. It would, of course, be possible to argue that this indicates that the 2,000-word AM is too small for these tasks. Such an argument should not be made in the context of this isolated routine, but in the context of an entire set of system processing functions. In such a context, this routine is a relatively minor consideration, since it is likely to be required only occasionally. In the remainder of this subsystem, only those routines identified as AD1, DD1, and CD1 will be discussed.

* This assumes that an attribute appears in roughly one-third of all records of the corresponding file. In the case of some attributes, distribution throughout the file is likely to be greater while for others it will be less. The figure "1/100" is therefore intended to be an estimated average.

The addition, deletion, or change of a directory entry is exactly the same as the addition, deletion, or change of data, with these exceptions: there can be more than one file directory entry in the trailer and the action table does not tell which entry is the one to be affected by the FMO; further, it is possible, in the case of the addition of an entry to the directory, that one must transfer a block of data in the directory to some other place in the directory in order to make room for the inserted entry. Deleting and changing directory entries, as opposed to deleting or changing data file entries, is a simple procedure of looping through the action time until all file directory entries have been checked. For this reason, these functions of ER2 are not discussed further. The comments of Paragraphs 6. 9. 1. 1 and 6. 9. 1. 2 can be seen to apply directly.

In adding data to a file directory, it may actually be necessary to read more data from disc in order that the trailer will eventually contain all directory entries that must be dealt with by AD1.

Without an actual system, it is very difficult to estimate the amount of data that must be read from disc to execute AD1. It is assumed that approximately 10 disc records will be required on the average, although this is perhaps a low estimate. In the event that 10 or more disc records must be read, GAP becomes relatively inefficient. The reason for this is that GAP must get data read from disc via the 1604-B memory. It is therefore evident that A3 is superior to GAP in efficiency for AD1, and, from Paragraph 6. 9. 1. 3, it can be inferred that it is therefore superior to the 1604-B only version of AD1.

6.9.3 ER3

The tasks performed by ER3 are concerned, primarily, with operations affecting logical names. Conceptually, these tasks appear to be relatively complex since logical names are the means whereby the entire data base is linked and referenced. Hence, the operations of ER3 affect more than one file directory, the D-directory, and, often, data records themselves. Despite the relative complexity of these operations, the sea surveillance ER3 routines function almost exclusively as "traffic cops," transferring program control to various components of ER1 and ER2. The only unique maintenance operation performed is with respect to adding, deleting, or changing entries in the D-directory. All other maintenance operations are performed using ER1 and ER2 components to effect changes in the corresponding directory and data record elements.

The subroutines of ER3 that perform operations on the D-directory have the prefix DDX in their name and are:

(1) DDXC — change D-directory entry

(2) DDXA — add entry to D-directory

(3) DDXD — delete entry from D-directory

As in other ER routines, DDXC, the "change" is interpreted as a delete (DDXD) followed by an add (DDXA). Hence, it is not necessary to study DDXC in any great detail.

6.9.3.1 DDXD. This routine which deletes an entry from the D-directory, proceeds exactly the same as DD2 (see Paragraph 6.9.2), except that the location of the entry to be deleted must be accomplished through the use of VLLU (see Paragraph 7.2.1.1). This is because the external name is a variable length field. However, the use of VLLU does not change comments made about deletion in Paragraphs 6.9.1 and 6.9.2.

6.9.3.2 DDXA. The addition of an entry to the D-directory is precisely parallel to the addition of an entry to any file directory, except that the D-directory is affected. Hence, DDXA is not ever flowcharted for the system.

6.9.3.3 Combination of Components into CLN, ED, and DX. This paragraph describes the way in which CLN, ED, and DX are constructed from DDXD, DDXA, and DDXC.

CLN, which changes (or reassigns) logical names, proceeds as follows:

(1) Change logical name in data record, using the action table to locate the logical name. Set A = 1 for this record.

(2) Change the logical name in the file directory, using CD1 of ER2.

(3) If the logical name does not belong to the static ship file, change the static ship directory entry, using CD1 of ER2 and skip to step (5). Otherwise, go to step (4).

(4) If the entry is a static ship file entry, use CD1 of ER2 to change every file directory entry, including the static ship entry.

(5) Use DDXC to change the D-directory.

ED, which enters a term into the D-directory, is simply an application of DDXA. No directories or records are affected by ED except the D-directory. This reflects a decision to permit the D-directory to contain synonyms (i.e., two different external names referring, via the D-directory, to one internal logical name).

DX, which deletes a logical name from the system,proceeds as follows:

(1) Use ERDR to delete the named record.

(2) Use DD1 to delete the file directory entry, and skip to step (4) unless the logical name is the name of a static ship file entry.

(3) If the entry is the name of a static ship file entry, use CD1 to delete all links to the static ship directory (seven additional uses of CD1).

(4) Use DDXD to delete the D-directory entry.

6.10 RELATED TRANSACTION ROUTINE

Basically, the Related Transaction Routine is a table driven routine. It determines what FMOs might be required to be generated as the result of applying one FMO. The tables would be user generated and would reflect, essentially, decisions the user made about the data interrelatedness and the procedures he would employ in file maintenance. It should be understood that the Related Transaction Routine is not actually needed in theory, but is likely to be used in practice. For example, in theory, a group of FMOs can be written to adjust the dynamic ship file whenever a ship has entered port. In such an event, however, the user is forced to attend to all problems of file relatedness, most of which could be automatically handled.

The logic of the Related Transaction Routine is presented in Figure 6-1. The routine first determines what actions may be required as a result of executing an FMO. For example, it determines that when an FMO has instructed the system to put a ship into port, that ship cannot have a non-zero SOA. The routine next checks to see if the ship's SOA is zero. If not zero, it issues an FMO to change the SOA to zero. If zero, it moves on to any other related transactions it must consider.

Despite the fact that the Related Transaction Routine is table driven, little use can be made of an associative memory unless many related transactions are stipulated by a user. For the file maintenance system assumes that after the Assembler has been executed, the entire augmented trailer is in associative memory if space is available. Inserting a related transaction table therefore runs the risk of destroying AM store and causing it to be reloaded for execution of the next FMO. For this reason, the Related Transaction Routine need not be studied further.

6.11 FAMILY CONTROL AND WRITE

The Family Control Routine simply checks the stack of FMOs until all FMOs of the family have been performed. Once they have, it consults the action table and for all entries whose A value is 1, it writes them onto disc. The only comment relevant to the associative memory system is that A3, by virtue of not having to transfer data into the 1604 prior to writing onto disc, will perform more efficiently than either A2 or GAP. A2 and GAP must transfer data to the 1604, and pay a penalty of an additional 7.1 microseconds per data word transferred. In addition, core storage must be provided for the disc write routine.

SECTION VII. QUERY PROCESSING

7.1 INTRODUCTION

The main purpose of a sea surveillance system is to retrieve and present stored information and/or values of a function with stored information arguments. The commands to retrieve and present this information are termed queries. The queries considered in this study (Paragraph 3.5) do not require all 108 attributes found in the data base (Paragraph 4.5.3). This does not mean that attributes not required should be eliminated; rather, it implies that the queries are typical and representative of a complete set which may require the entire data base. In like manner, the data base given may be considered as representing the required data base for some time interval during its development and use. Thus, the data base should not be considered in final form.

7.2 USER'S QUERY LANGUAGE

There exist several methods to admit query processing. At one extreme, the processing required for each query may be completely encoded in machine code, stored on the system's tape, and retrieved by a call. This call (perhaps the query's name or number) may be considered as a query language. Since neither the data base nor the queries may be considered complete at any one time, it is impossible, or at least time-cost prohibitive, to follow this possible method of effecting queries. Examination of the given queries to determine some general features shows that each query requests statistics about, or descriptors of, a set of attributes of one or more items. The set of attributes and/or items are assumed to satisfy some conditional statement. It is felt that the techniques needed to directly supply answers to each query are not explicitly known at this time. The user of the system must be relied upon to state the query in a permissible way to retrieve the pertinent data and then interpret the results.

A user's query language is one which attempts to provide a strong linguistic capability together with convenience of use. Such languages are either translated into machine code or a command list for processing by a compiler type pre-processor. Additionally, such languages may be regarded as parameters controlling processing as by an interpreter. In this study it was decided to translate a user's query language into an internal language in Polish prefix form. This internal form is then "compiled" as object code by a pre-processor (Paragraph 7.3) into a command list form. A Run Routine (Paragraph 7.4) then "performs" each element of this command list in an interpretive fashion. The Run Routine is controlled by an executive level routine which interleaves input-output (Paragraph 7.6) requirements with portions of the Run Routine. The executive level system is called the Controller (Paragraph 7.5).

It is beyond the scope of this study to formulate a user's query language which is amenable to associative memory processing. Also, it is deemed inappropriate to consider translation of an existing user's query language by use of an associative memory since the existing language was formulated to be translated by non-associative memory processing. The only restriction placed on the user's language is that it can be processed into Polish prefix form.

7.2.1 User's Query Language Translator

Some of the tasks that the translator must perform are:

(1) Resolve all synonyms into one term.

(2) Eliminate redundant and "noise" words.

(3) Resolve functions and operations into canonical form.

It seems that one pass of the input expression in user's query language is sufficient to accomplish these (and perhaps other) tasks as follows:

(1) An input name (an operation to be performed, an attribute, a descriptor, etc.) is looked up in the dictionary. Unadmissible operations, attributes (and the associated descriptors), comments, etc., are eliminated as redundant or "noise" words.

(2) All resulting attribute and descriptor pairs are compared with the D-directory. (If a descriptor is not specified for an attribute, it implies that all descriptors for that attribute are referenced.) All descriptor synonyms are resolved into one word. If the descriptor is an item's name, it is resolved into a logical name.

(3) The resulting resolved and "clean" expression is then translated into Polish prefix form.

7.2.1.1 Equating Synonyms and Logical Names. As may be expected, the associative memory proves useful for synonym equating, item descriptor to logical name replacement, and, to a lesser degree, for translating operations and relations into canonical form. The heart of both operations in the associative memory is the variable length field look-up routine VLLU (see Appendix C). This routine (or macro operation) considers as input a set of words, W1, containing the name or expression to be looked up either in the D-directory or the dictionary, a set oı words, W2, containing masks to be applied to W1, and of course, either the dictionary or the D-directory.

Note that the AM sequence relies heavily on the ability to shift buffers and "AND" results at step (5) of VLLU, and that an AM index register is used to control stepping down data arrays W1 and W2. In almost all our experimenting to date these features of the AM have been useful. Further, the "micro operations" provide important initialization in this case, and, surprisingly, they tend to take important roles in other routines as well.

Of several methods for developing VLLU, two are especially worthy of notice. Both method 1 and method 2 employ one sequence of AM programming to look up a variable length field for a particular arrangement of W1 and W2.

Versions 1 and 2 differ in the way in which shifting is performed. Specifically, the shiftings of W1 and W2 are performed after each performance of sequence S, in version 1, while the shifting is performed before entering sequence 8, in order to generate all cases for version 2.

The important difference between version 1 and version 2 is that version 1 employs a subroutine (shifter) to generate the appropriate arrangement of W1 and W2. This subroutine is effectively performed by 1604-B coding, and, hence, version 1 mixes 1604 and AM coding. Version 2 employs 1604-B coding, if at all, only to set up the full range of permutations of W1 and W2 before entering AM coding and does not leave AM coding for 1604-B processing until (if at all) it is through with AM processing.

If it is assumed that one is interested in items whose length can be expressed in characters, version 1 halts AM processing eight times and initiates AM processing eight times. Version 2, on the other hand, initiates AM processing once and halts it at most once. It can be expected, therefore, that version 2 is the faster of the two, although it requires more 1604-B storage for execution. In surprisingly many cases this situation will exist, and the programmer interested in minimizing speed will select a program which segments responsibility between the AM and 1604 in a manner similar to that of version 2.

7.2.1.2 <u>Polish Prefix (Polisher).</u> At this time, no advantages have been found for any Polish prefix routine (Polisher) to functions using the AM exclusively. In some respects this is surprising. The AM has a meaningful potential for efficiently discovering and flagging parentheses, in parallel. This is one way of manually polishing an expression.

A subexpression is defined as of order 0 if there are no parentheses within the parentheses marking its beginning and end in fully parenthesized form. Thus, (pvq) and (a+b) are of order 0. An expression is of order n+1, within its delimiting parentheses, the highest ordered expression is of order n, and the expression itself is not of order n.

A manual procedure for polishing a fully parenthesized, well-formed expression is as follows:

(1) Set order of test expressions = 0.

(2) Find all expressions of order of test.

(3) Polish all such expressions.

(4) If there are any parentheses remaining, continue, otherwise the expression is polished.

(5) Step order of test +1, and go to (2).

For example, given

$((a \vee (b \wedge c)) \wedge (-(a)))$

Iteration 1 = : $((a \vee \wedge bc) \wedge (-a))$

Iteration 2 = : $(\vee a \wedge bc \wedge -a)$

Iteration 3 = : $\wedge \vee a \wedge bc - a$

or, in more conventional Polish notation,

Iteration 3 = : AU aAbcNa

The routine to accomplish this polishing may be broken into two parts. The associative memory in the Goodyear Associative processor 1604-B configuration may be used to advantage in the second step; that is, finding parenthesis pairs for successive orders (0, 1, 2, ...). The third step, "Polish all such expressions," is more readily accomplished in the 1604-B. The ideal configuration of associative memory, general purpose computer for this technique is the A3 1604-B configuration. It is well-known that Polish prefix processing may be accomplished by a general-purpose computer.

7.2.1.3 DeMorgan's Law and Double Negation Law

(1) Introduction. In many retrieval systems, a Boolean criterion for retrieval of data, such as "$p \vee (q \wedge r)$" is applied to successive elements of the data base. In the sea surveillance system set theoretical operations are performed, whenever possible, on directories. In the example given, three lists are generated, one containing those directory elements that are in the class p and the others containing those which are q and r. The intersection of the q and r lists is formed, and the union of this intersection with p is then generated. The resulting list is a directory for those items that will respond to the query "$p \vee (q \wedge r)$."

It should be noted that it is not always possible to respond to a Boolean query in this way. For example, it is not advisable to generate directories that can respond to certain relational tests, such as "within x miles of y" or "closest in distance to z." On the other hand, those Boolean operations that can be performed will often reduce the number of times one must go to the data itself in order to perform the required calculations.

One Boolean operator, nevertheless, causes some trouble. "Not x" is to be interpreted as "the class of all objects except those which are x". To minimize the number of directories that must be examined, and to minimize references to actual data, Boolean laws applying to "not" are employed to optimize processing. The two laws of greatest importance are De Morgan's Law and the Law of Double Negation.

(2) Conventional Memory Algorithm DMDN. An algorithm for conventional memory processing of an expression applying both laws is given to illustrate procedures. This algorithm is based on this equivalence: ($NNp \equiv p$) for the Double Negation Law and the definition of an "elementary expression" for De Morgan's Law.

An "elementary expression" depends upon the Boolean operators available in the query language. The algorithm is designed for a language containing these operations: N, negation; A, inclusive alternation; and K, conjunction. (It will work for a language containing other operators.) In some user's query language, however, an extended definition of DeMorgan's theorem can lead to a different concept of an elementary expression. De Morgan's theorem employs the following equivalences

$$\overline{(p \vee q)} \equiv \bar{p} \cdot \bar{q}$$

$$\overline{(p \cdot q)} \equiv \bar{p} \vee \bar{q}$$

An extended list of equivalence can include a selection of the following:

$$\overline{(p \supset q)} \equiv \bar{p} \not\subset \bar{q}$$

$$\overline{(p \equiv q)} \equiv \bar{p} \not\equiv \bar{q}$$

$$\overline{(p \not\subset q)} \equiv \bar{p} \supset \bar{q}$$

$$\overline{(p \not\equiv q)} \equiv \bar{p} \equiv \bar{q}$$

With appropriate selection from the Polish operators,

C	(conditional)
E	(biconditional)
R	(exclusive or, non equivalence)
X	(converse non-conditional or converse non-implication)

noting that C and X are dual, as are E and R, De Morgan's theorem can be generalized by defining the appropriate operators and expanding step (3) of the DMDN algorithm which substitutes the dual operator.

The algorithm assumes that the input is a well-formed expression in Polish prefix form stored in a list storage area, L. The output is also in Polish prefix form and also stored in list storage area, L. The DMDN algorithm follows:

(a) The input expression is scanned termwise until a negation operator "N" is encountered.

(b) When an "N" is encountered, the successor of the "N" is examined.

  (i) If it is an elementary expression, the scan continues searching for an "N" (step (a).

  (ii) If it is an "N", both "N"s are deleted (Double Negation Law) and the latter parts of the expression are all popped up two places in the list. The scan resumes searching for an "N" at the place where the first "N" was encountered (step (a).

  (iii) Otherwise, go to step (c).

(c) Substitute the dual of the operator for the N preceding the operator (De Morgan's Law).

(d) Substitute N for the operator.

(e) Scan forward for the second term of the operator just dualized. Push that term and all subsequent portions of the list down one place (Figure 7-1); write an "N" in the opening created.

(f) Resume scanning at (a), except that the scanning of the expression is already complete up to and including the dualized operator, so that only the parts of the expression after it need be examined.

(g) Terminating Condition. The procedure terminates once a complete scan of the expression has been made.

The subroutine which searches for the pushdown point (step (e) in this algorithm exploits the following features of the Boolean expressions in Polish prefix form.

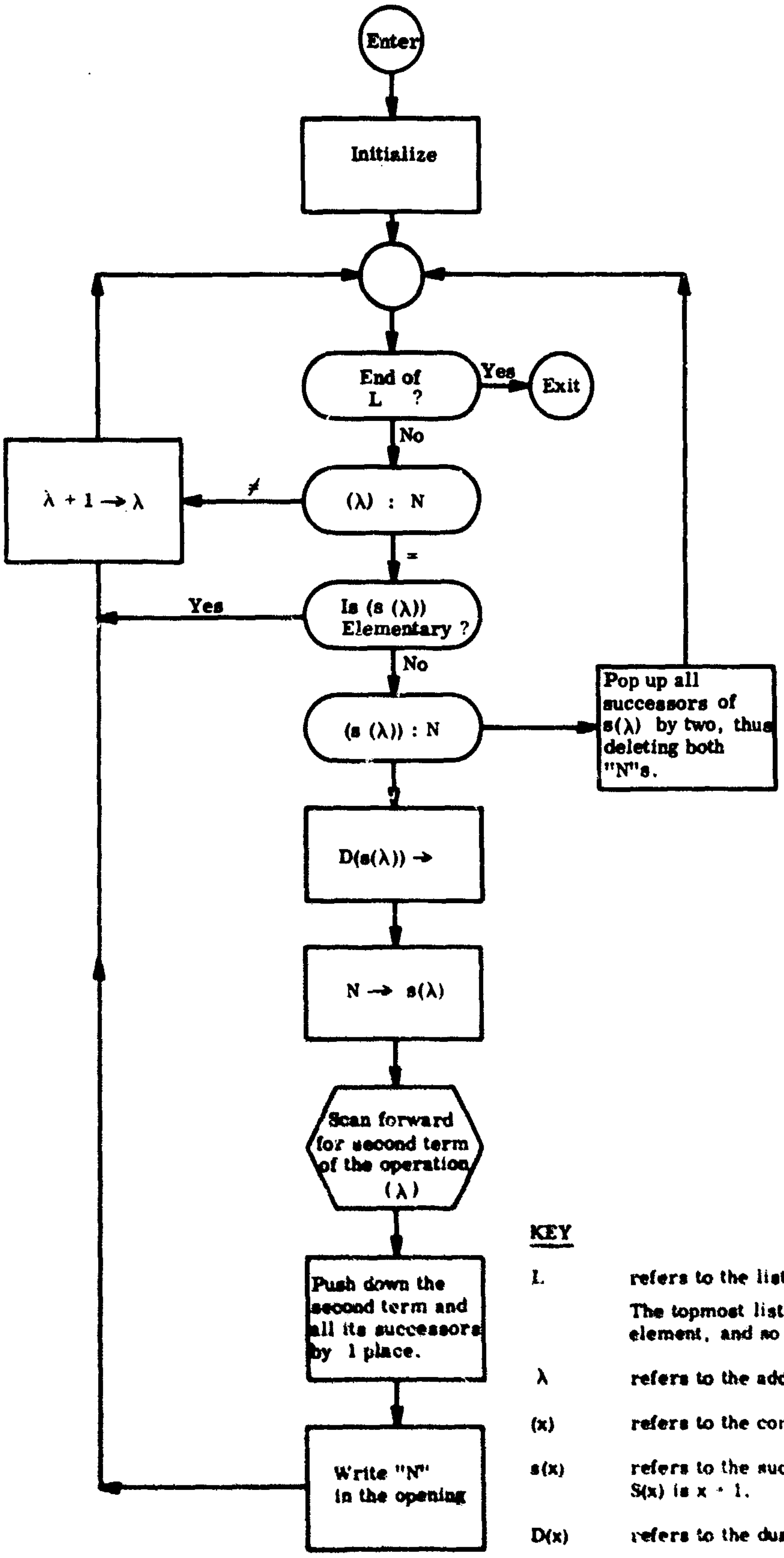
(a) N can be ignored in favor of counting expressions, since when "Nx" is any expression, determining the end of "Nx" is equivalent to determining the end of "x."

(b) Operators U, A, C. E, R, and X have two variables. The pushdown point is immediately following the first variable when the governing operator is one of these.

(c) The operators U, A, C, E, R, and X, when nested, signal the beginning of expressions within expressions. If, during a scan we encounter one of them, we can increment a counter of the number of expression we have begun. Every time we end an expression, we can decrement that counter. Such a counter can then signal the end of a nested expression. Our pushdown point is therefore detected.

The method is flowcharted (Figure 7-1) as a scan for the pushdown point. K is an address and K + 1 and all other addresses in L after K must be pushed down one place to make way for the insertion of the N. $(K) \in \{E\}$ is a statement that the content of K is a member of the set of elementary expressions. Branching is done depending on the truth or falsity of that statement. The address of the element being scanned, λ, is the only parameter, other than the list itself, required by the subroutine.

(3) Double Negation Algorithms. It can be seen from the DMDN algorithm that the Law of Double Negation can be performed by the Goodyear associative memory 1604-B in several ways. However, two algorithms will be given, called DN1 and DN2, respectively. The actual encoding and flowcharts for these algorithms are also


Enter
Initialize
End of L ?
Yes
Exit
No
(λ) : N
≠
λ + 1 → λ
=
Is (s (λ)) Elementary ?
Yes
No
(s (λ)) : N
Pop up all successors of s(λ) by two, thus deleting both "N"s.
D(s(λ)) →
N → s(λ)
Scan forward for second term of the operation (λ)
Push down the second term and all its successors by 1 place.
Write "N" in the opening

KEY

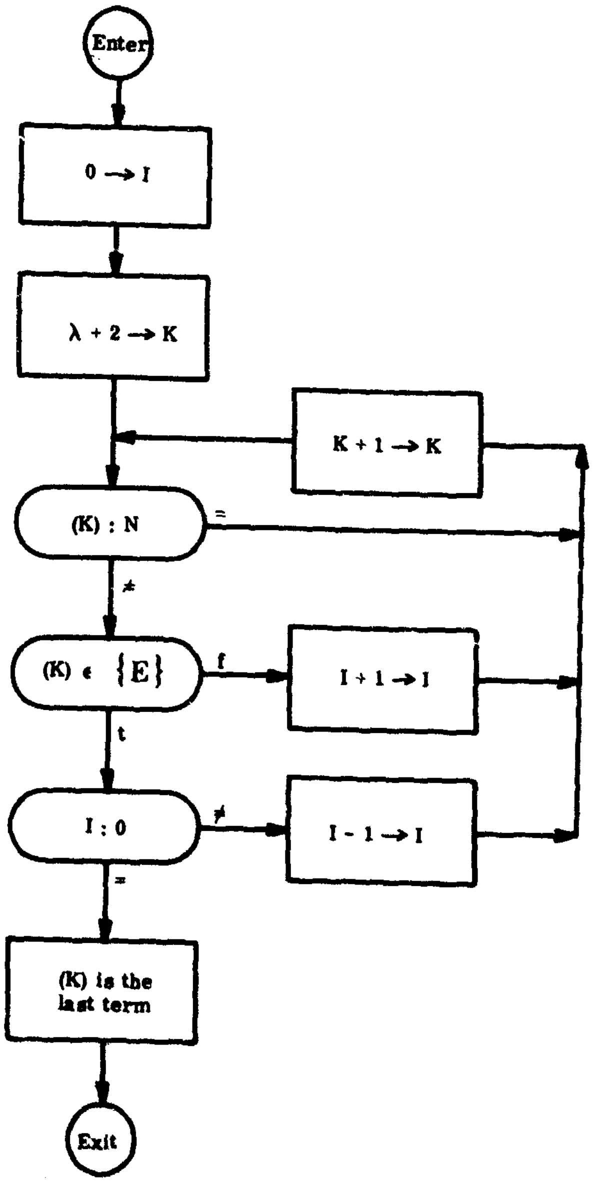
L refers to the list representation of the expression. The topmost list element is the leftmost expression element, and so forth.

λ refers to the address of the element being scanned.

(x) refers to the contents of x, where x is an address.

s(x) refers to the successor of x. Where x is an address, S(x) is x + 1.

D(x) refers to the dual of x, where x is an operator.



ation of the expression.
the leftmost expression

element being scanned.
where x is an address.
. Where x is an address.

x is an operator.

Table of Duals

The dual of	is
A	K
K	A
C	X
E	R
R	E
X	C

Figure 7-1. Scan for Pushdown Point Flow Chart

2

given (Figures 7-2 and 7-3). Of the two algorithms, the second, which most nearly takes advantage of the parallelism of the associative memory, is most efficient.

Algorithm, DN1 (Figure 7-2) assumes as input, a well-formed Polish prefix form. It follows:

(a) The symbols are loaded, sequentially, into the AM.

(b) The EMC instruction sets response store for all "nots" in the expression, and the buffers are shifted down one location.

(c) An EMC instruction sets response store for all "nots" in the expression, and the results are "AND"-ed into the buffer.

(d) If there are no responders the routine exits; otherwise, it continues.

(e) The contents of the first responder are read.

(f) The portion of the responder is used to set up a transfer that will transfer a 1604-B token substring from $\lambda + 1$ into AM location $\lambda - 1$ through the end of the string.

(g) The transfer is accomplished, and control is given to step (2) of the algorithm.

There are three interesting features of the DN1 algorithm:

(a) It is, from the point of view of the AM, an inefficient algorithm. The two EMC steps can set multiple responders, but the algorithm processes only one responder at a time. A natural suggestion for improving the algorithm would be the use of an RCR instruction at $\lambda = 8$ instead of the RCF. An algorithm exploring this idea has been coded, but shows less improvement over the method than that of version 2.

(b) The sequence of steps, 11, 12, 13, i.e.,

```
LDI
DEI
SIX
```

it required to obtain the value of R for the LDR instruction, and it brings home forcefully the limitation of the AM repertoire or instructions for elementary control arithmetic. It might be asked why a 1604-B instruction, such as an RSO, was not used to obtain a value of R equal to $\lambda - 1$. The answer is to minimize total time. In order to employ the 1604-B instruction a HLT is required, followed by a few set-up steps

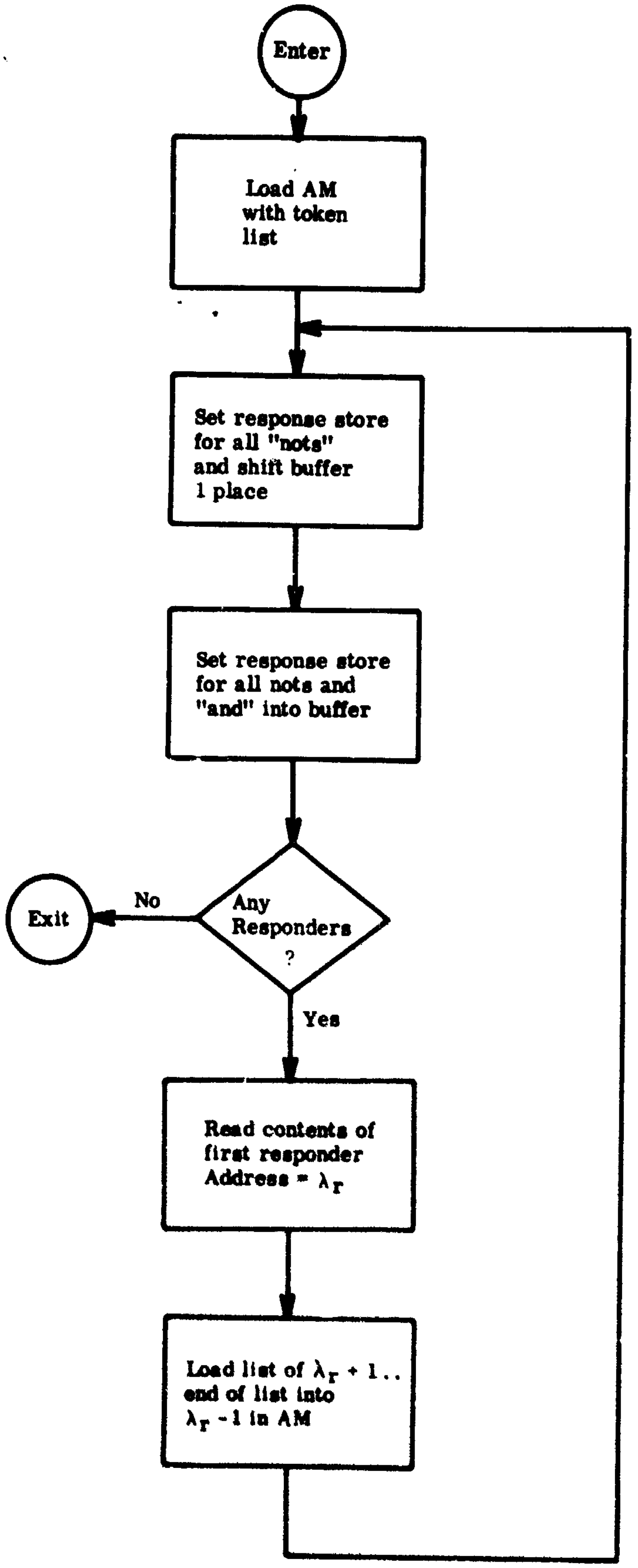

Enter
Load AM with token list
Set response store for all "nots" and shift buffer 1 place
Set response store for all nots and "and" into buffer
Any Responders ?
No
Exit
Yes
Read contents of first responder Address = λ_r
Load list of $\lambda_r + 1$.. end of list into $\lambda_r - 1$ in AM

Figure 7-2. Double Negation DN1 - AM Form

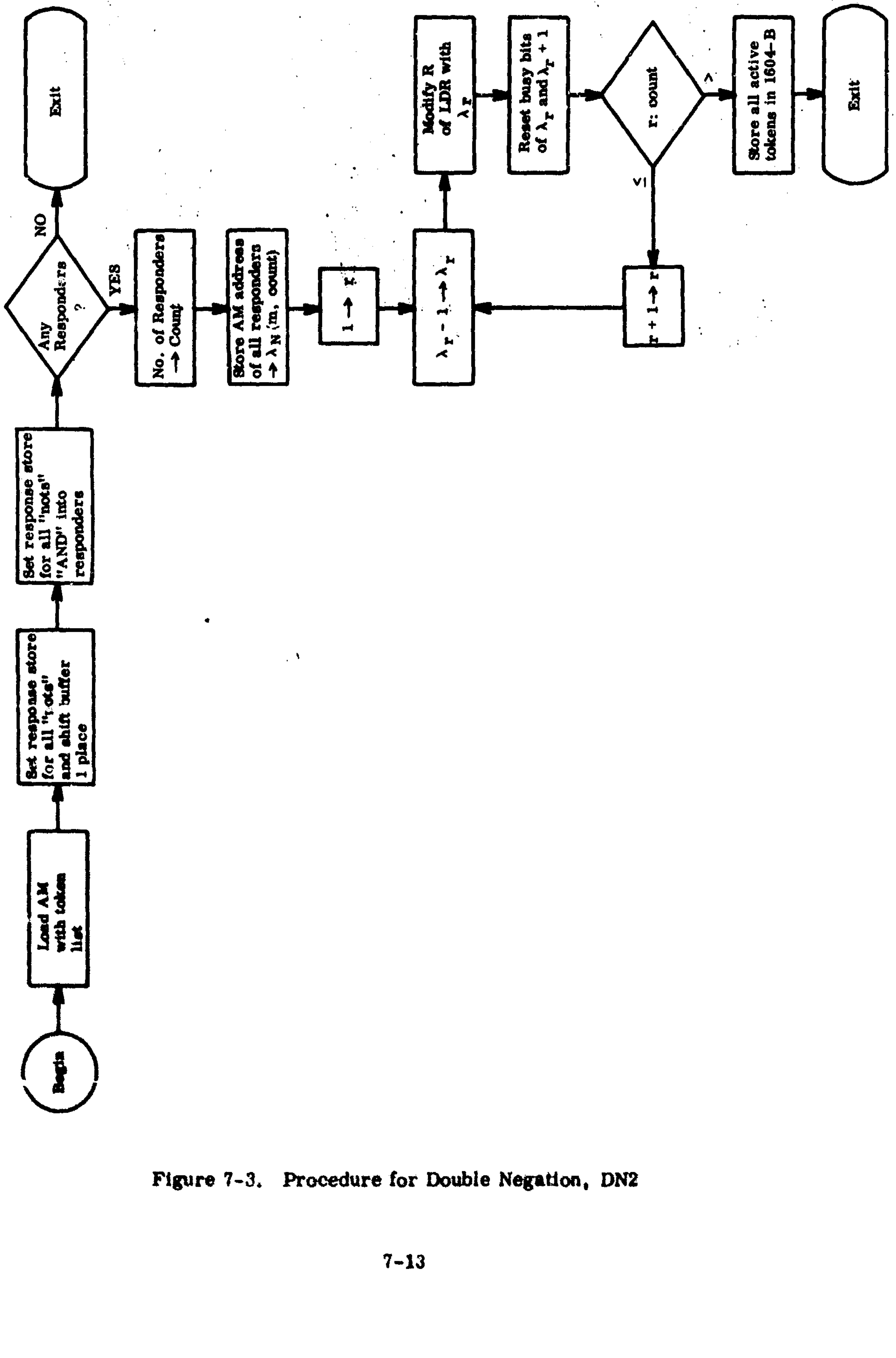

Begin
Load AM with token list
Set response store for all "nots" and shift buffer 1 place
Set response store for all "nots" "AND" into responders
Any Responders ?
NO
Exit
YES
No. of Responders → Count
Store AM address of all responders → λN (m, count)
1 → r
λr - 1 → λr
Modify R of LDR with λr
Reset busy bits of λr and λr + 1
r : count
>
≤
r + 1 → r
Store all active tokens in 1604-B
Exit

Figure 7-3. Procedure for Double Negation, DN2

and a "Force or Resume." Getting out of and into the AM coding sequence is clearly more costly than the slight inefficiency resulting from the selected code.

(c) The shifting of the buffers, followed by "AND"ing, proves helpful.

(The second algorithm for double negation, DN2 (Figure 7-3) uses the "busy bit" stored with each word in the AM as a means of erasing data, in this case, "nots." When the results of DN2 are required, they can be read out with a "random unload monitor (RUM)."

The Lawof Double Negation would be easy to apply in GAP if the D and E buffers could be shifted in both directions. Assume that the AM contains the expression to which DN is to be applied. Then the following DN3 algorithm would do the job (useful in the AZ):

(1) Set response for all "nots."

(2) Shift response down (up) one place.

(3) Set response for all "nots" and "AND" the results.

(4) Erase all responders.

(5) Shift response up (down) one place.

(6) Erase all responders.

The DN3 algorithm shows a truly parallel symbol manipulation operation, and is one reason that we believe the AM's symbol manipulation capability should be explored.

This "double shift" algorithm, or more properly, the A2 algorithm for double negation points up one criticism of the Goodyear associative memory; that is, its inability to shift the D and E buffers in both directions. Another criticism resulting from looking at DN2 is that the R field of the LDR instruction is very hard to set as a function of the results of the AM program

(4) De Morgan's Law. This section has presented an algorithm, DMDN, designed to simultaneously apply Double Negation and De Morgan's Law. This algorithm was given for a conventional memory processor, or general-purpose computer, to show procedures. Two algorithms (DN1 and DN2) and their coding were given for the Goodyear Associative Memory 1604-B configuration for double negation. Additionally, a third algorithm, DN3,was given for double negation in the A2 1604-B configuration. At this point, it remains to specify an algorithm for applying De Morgan's Law in an associative memory processor.

This algorithm, DM, assumes an input statement in Polish prefix form as follows:

(a) Flag all non-"not" operators preceded by "not." If there are none, exit.

(b) Dualize all flagged operators.

(c) Interchange the relative position of the flagged operators with the "nots" preceding them.

(d) Search out all breakpoints.

(e) Make an opening at the breakpoints and write a "not" in each.

(f) Return to step (a).

Step (a) requires five AM instructions:

LDR	Set up for EMC
EMC	Set responders for "not," shift after.
LDR	Set for EMC
EMC	Flag all non-not operators and "AND" the results
JNR	Exit if no responders.

Step (b) can be performed in the AM very easily.

Step (c) can be performed in the AM, but once again the capability to shift buffers in both directions would be helpful.

Step (d) can be competitively performed in the AM. The more breakpoints that one must find, the more competitive it becomes.

Step (e) is time consuming in the AM while step (f) is trivial.

7.2.2 General Comments

The AM can be expected to be more helpful in evaluating queries, as opposed to manipulating them. In fact, the interpretation of Boolean expressions with an AM is extremely natural. The searches, EMC, LTC, GTC, etc., generate subsets of a set. The response store gives us a Boolean array of members of the set, the "read address of responders" instruction can be used to give absolute location in the AM or, more helpful in some cases, relative locations in the 1604. The "read responders" give us the capability to get the entire subset. In dictionary and directory lookups, "masked read of contents of responders" can be employed to extract "pointers" from the directories or dictionaries.

Whether the full potentialof the AM for interpreting the internal query language will be realized by the user is still an open question, since the problem may prove to be "input-output bound," and the potential of the AM may not result in any user benefit.

7.3 QUERY PRE-PROCESSOR

7.3.1 Introduction

The query pre-processor accepts as input a query given in Polish prefix form (ppff) which has been processed by the translator;

- all synonyms have been resolved,
- all redundant and "noise" names have been eliminated, and
- the ppff input statement is in canonical form.

The purpose of the pre-processor is to form a command list λ as its output. If desired, an analogy may be drawn between source code that is pre-processed into object code by compilation techniques. The pre-processor may be thought of as a compiler operating upon an input statement in ppff (source code) to produce a command list (object code). The command list cannot be performed directly; it is performed interpretively by a Run Routine (Paragraph 7.4) under control of the Controller (Paragraph 7.5).

7.3.2 Input Polish Prefix Form

Polish prefix form may be defined as a sequential list of elements where each element consists of an operation ϕ and a certain number of variables. Each operation requires a certain number of input variables. Call this number the rank of the operation and denote it by ϕ (m). For example, if the operation, ϕ corresponded to "Distance," two variables are required. Thus,the rank for this operation is 2. Again, if the operation were "maximum," the rank would be one, since only one list is required as input (even though the list may contain many entries). The format of ppff may explicitly or implicitly give the variables for an operation. If implicit, the last variable stored as a pushdown (last in,first out) list is the variable to be used for an operation where an insufficient number of variables is explicitly expressed to satisfy the order of the operation. If K represents the conjunction ("and") operation, A the inclusive disjunction ("or"), and N complementation, while lower case alphabetics denote variables, then two examples of ppff are UAaNbUbNa and UAabUNaNb. The ranks of U and A are 2 and N is 1.

If a variable is implicit for an operation, that is, it is stored and must be retrieved, the operation is said to <u>range</u> over the operation which generated the required variable. For example, the last V (left-most) operation performed in the example ranges over all preceding operations. Thus, the range of an operation gives an indication of storage requirements. Let the number of operations, k, that a particular operation ranges over be noted as θ (k).

It is anticipated that the variables of the operation in a query statement will be lists of logical names. The <u>a priori</u> operations may be examined along with the inputs (variable lists) and some decision made as to the maximum length of the output list. Let the length of each list be known as the <u>norm</u> of the variable. The output can take on one of four values: $(N_1 + N_2)$, min (N_1, N_2), max (N_1, N_2) and $(N_1 - N_2)$, where N_1 and N_2 are the norms of the variables. Indeed if the resulting norm is zero, a great deal of processing time is saved (since the nor computed, <u>a priori</u>, is a maximum valued norm). Thus, it is possible to compute the norm of the results of every operation. Since this output may function as an input variable for a successive operation, it is possible to compute the norm for the entire query. If, as suggested, the norm of the query were zero, the query would be satisfied, <u>a priori</u>; that is, no data exists in the data base to answer the norm. Thus, a great deal of Run Routine time is saved.

Certain assumptions regarding the contents of the input ppff list may be made; in other words, the specifications of some of the contents follow:

(1) Each element is identifiable as an operation or a variable.

(2) Since each element was processed through the D-directory by the Translator, the following information is included:

(a) An operation,

- θ, (item (2), Paragraph 7.3.3.3)
- θ (m), the rank of the operation.

(b) A variable,

- $\left\{A_v\right\}$ or ¢, (item (3), Paragraph 7.3.3.3)

- N, (item (3) (f), Paragraph 7.3.3.3)
- L, (item (3) (d), Paragraph 7.3.3.3)

(3) An operation is given after (right to left) its variables.

7.3.3 Output Command List

7.3.3.1 Variable Specification. Every element λ i of the command list is associated with one and only one operator θ_i given in the input list which is ppff. Associated with each operator are one, two, . . . variables, depending upon the rank of the operator. To speak of processing a ppff canonical form statement, it is necessary to determine an ordering of variables and operators. Properly, such a statement is "read" from right to left; then, an element to the left of another element is said to succeed it. It is clear that some operations use as input variables (implicit variables discussed earlier) the output of a preceding operation. It is necessary in forming the command list to explicitly associate with each operation the set (the number is given by the order of the operation) of variables that it ranks over. This is one of the tasks that the pre-processor accomplishes. Simply, the rule applied is this: "Whenever an operation is found in the input list which has a rank equal to the number of variables preceding it, such operation is said to be "rank satisfied." If such is not the case (the rank then must be greater than the number of preceding variables or else canonical form does not hold), a sufficient number of preceding variables is associated with the operation until it is "rank satisifed." The added variables are "implicit variables."

7.3.3.2 Order of Performing Operations. From the preceding example, it should be clear that the operation involving implicit variables (that is, the output of preceding operations) must be performed after the operations which form the implicit variables.

The input list implies an order of performing operations. This fact combined with the observation in the preceding paragraph yields this rule:

"An operation of range k may not be performed until all preceding operations of lesser range have been performed."

This rule suggests that the operations in the command list could be sorted on their range. But this does not take into account the word "preceding." Suppose that all operations were ordered on range and further suppose that when an operation was performed, its output was stored in the "input location" of the proper succeeding operation. If all variables of the succeeding operation were formed, a priori, it could be performed before others that are not in its range, but with a lower range number. This particular aspect of operation ordering is detailed further in the Controller (Paragraph 7.5).

Another observation of the input list is that all operations of the same range may be performed in an arbitrary order. This will be somewhat changed to order those variables requiring the fewer files (directories) first and those requiring the data files last.

The situation is similar to a CPM (critical path method) chart. There exists at any one time a series of operations which may be performed simultaneously (provided that multiprocessing capabilities existed). In serial processing, this amounts to the statement that the processing order for this set of operations is arbitrary. Then, the switches (α, β, γ) in Figure 7-4 will order these operations such that those with a (¢) (item (3)(b) in Paragraph 7.3.3.3) are performed at their "latest start time" and those without a (¢) at their "earliest start time." Thus, operations with (¢) variables are delayed as long as possible. The purpose of this procedure is to retrieve as little data from the data records (or read as few data records) as possible by "narrowing" in on the required set, a priori, as much as possible.

7.3.3.3 Elements of the Command List. Every element λ_i ($1 \leq i \leq L$) has this form (which is called λ canonical); B, ●, V; where:

(1) B = α, the operation may be performed, data ready.
= β, the operation may not be performed, data not ready.
= γ, the operation has been performed.

(2) ● is the machine code transfer instruction (or some similar device) to the beginning instruction to effect the desired operation, followed by:

(a) # (●), the norm of the operation; that is, the maximum output size.

(b) WS_t, the beginning address to store the output.

(3) V may be one or two similar elements; that is, λ_i B●V or B●VV depending upon the rank of the operation ●. Each V consists of V, A, W, L, M, N, where:

(a) V is either an attribute identifier or logical name.

(b) A = $\{A\}$, the D-directory address of the set of file directory addresses if V is a logical name, or

= ¢, if the data must be obtained from the data files.

= O, if the desired data is already in memory.

(c) W = WS_t, the beginning address where the data is:

(i) stored, if A = O, or

(ii) to be stored, if A = $\{A_v\}$ or ¢.

(d) L = L_1 or L_2 to specify θ input.

(e) M = +, if the data in WSt must be saved for a future θ or
= *, if the data in WS_t need not be saved for a future θ.

(f) N = # (v), the norm of the variable. A norm of n implies 32n words must be set aside for storage.

Contrasting an element of this λ with the contents of the input ppff statement, the elements that must be provided by the pre-processor may be developed.

7.3.4 Query Pre-Processor

7.3.4.1 Required Outputs. From the input and output forms the following elements must be generated or developed by the pre-processor.

(1) Explicit variables for implicit variables.

(2) Assigning indexes to storage areas such that they cross reference as required.

(3) Indication of disposition (item (3)(e), Paragraph 7.3.3.3) of inputs for an operation.

(4) Norms of operations.

(5) Rank of operation.

(6) Canonical form of the command list.

7.3.4.2 Pre-Processor Algorithm. Two methods of documenting and effecting the pre-processor will be presented. The first method, presented in this section is an algorithm tailored expressly for the Goodyear Associative Processor 1604-B configuration. The second method is a flowchart tailored for a conventional (non-associative memory) general-purpose computer. Since an algorithm and a flowchart are both a method of problem analysis, it is hoped in this approach to obtain some indication of the impact of an associative memory on problem analysis.

Assume that the entire ppff input query is stored in the associative memory (as output from the translator) such that the statement is read from left to right as memory addresses increase. Further, assume that each operation and variable have those elements associated with them as given in Paragraph 7. 3. 2 (contents of the input list) in a fixed field format with one element per associative memory word. Let there be R operations and a total of Q entries — operations, variables, and their associated elements in the list.

It was shown earlier that processing time is decreased by proportionately decreasing interleaved 1604-B and AM processing. In other words, accomplish the maximum amount of processing on the input while it is in one memory device before transferring it to the other memory for additional processing. The algorithm will attempt to follow this observation.

Assume that the entire query statement is in the associative memory. The scheme is to generate canonical form and to change implicit variables to explicit variables by reading from the associative memory into the 1604-B core memory and then back into AM with no 1604-B operations. The algorithm follows.

(1) Reset and search for operations in AM (V=O).

(2) Jump to step (14) on no responders.

(3) Read address (R) of first responder.

(4) Random block unload three (N) words from R into 1604-B address $h + q_2$, initially $q_2 = 1$.

(5) Increment index of previous instruction by $[2 + 6$ times the rank of the operation $(H + q_2 + 1)]$. Effectively, when step (4) is performed again, a proper number of spaces is available, explicitly for the proper number of variables — that is λ canonical form being generated.

(6) Erase first responder.

(7) Perform subroutine SPO with inputs

M = Search variables (V = O).

N = 2 (between addresses).

Θ = Beginning address of query list.

Θ = R of instruction 3.

(8) Jump to step 1 on no responders.

(9) Read address (K) of first responder.

(10) Random block unload six (N) words from K into 1604-B address h $+q_1$), initially $q_1 = 3$.

(11) Increment q_1 by 6.

(12) Erase first responder.

(13) Jump unconditionally to step (8).

(14) The entire query is now in the 1604-B. It is "close" to canonical form. No data has been added to the list; only "room ' has been made for it. Let the beginning 1604-B address of the list be B and the ending address be E. Form (B-E+1) and set equal to N.

(15) Erase all responders.

(16) Random block load into associative memory locations b to E'.

At this point, the range of each operation will be determined. Each operation that has an implicit variable ranges over at least one other operation. If the variable indicators in the associative memory could be examined at this time, only explicit variables would be found. (Room was made for the implicit variables in step (5) of the algorithm, but no variable indicator was inserted since this would be a 1604-B operation at that time.) A variable indicator is required for those variables before proceeding. Relative to the location (L) of each operation, this indicator occurs for rank = 1 in location (L + 4) and for ranks greater than 1 in location $[L + 4 + 6(r-1)]$. (What a convenience a buffer advance of n places as a micro operation would be if incorporated with the NO operation.)

In like manner, it is known that if L is the address of the succeeding operation, then these variables occur in position (L-6) constantly unless the address of the preceding operation is greater. These addresses provide a method for marking these variables. Let a mask M be formed with zeros only in bit positions which indicate a variable.

(17) Search for operations.

(18) Read address (K) of first responder into H.

(19) Load H into register k.

(20) Increment register k by 4 (for the first variable).

(21) Random block load one variable word indicator with mask M into location (0 + register k).

(22) Increment register k by 6.

(23) Erase first responder.

(24) Jump to step (28) on no responder.

(25) Read address (K) of first responder into H.

(26) Jump to step (19) on index high ($H \leq$ register k).

(27) Jump to step (21).

(28) To insert locations corresponding to operations, write into responders through a mask of zeros.

Prior to step (17), the range of an operation was starting to be computed. Consider, in this regard, a Polish prefix statement. If it is canonical, then there exists one and only one output list. Since every operation generates a list, such lists must be used as input variables for subsequent operations. The canonical form states in essence that if an operation requires an input list, it is the output list of the immediate predecessor operation. Then, as a programming tool, consider a pushdown list, where the elements entered are operation outputs (lists), such that the last element in is the first element out. The rule for using this list is:

> "If the variables of an operation are specified (see item (3)(b), Paragraph 7.3.3.3), then the range of the operation is one. Enter the operation (with its range) into the pushdown list (PDL). If an operation requires variables; that is, it contains implicit variables, retrieve as many operation outputs — variables — from PDL as required. The range of such an operation is greater by one than the maximum of the ranges given with the PDL retrieved operations. Enter this last into PDL and continue."

The range accumulators ● (k) occur after the operation code. Initially, all accumulators will be set to contain a one in the rightmost bit with all other bits zero. To increase the range, left shift the given range by one position. Since the range of a particular operation is equal to the maximum range plus one of all operations that the particular operation ranges over, the desired range may be formed by writing through a mask of zeros with a left shift of one into the range accumulator. The highest ordered bit is the desired range. Since relative range relationships are of interest, this processing is acceptable.

It is also required to indicate the input-output relationships between variables and operations. A tag may be formed corresponding to the AM address of each operation. This tag identifies the output.* Then, this tag (the AM address of the operation) is placed in the address portion, WS_t (item (3)(c), Paragraph 7.3.3.3), of the variable section of the operation which ranges over it. Several initial values used later must be set. It is less meaningful to indicate their initial settings until they are introduced. However, the next step will be construed to be the initialization of all subsequent values:

(29) Initialize.

(30) Search for operations.

(31) Advance buffers and write constant z** into all responders. This sets range of all operations to zero; some of the ranges are correct.

(32) Reset and search for operations (to <u>retard</u> buffer).

(33) Read addresses into a list beginning at 1604-B address P. Then read count of responders as N and write into responders through a mask to save the AM addresses of the operations. This will be used in tagging.

(34) Search for variables, all explicit and implicit.

(35) Read addresses into a list beginning at 1604-B address V.

(36) Using previous step, search for implicit variables only.

(37) Jump to step (55) on no responders.

* If the same operation with the same input variables is to be performed more than once, this assignment prevents substituting the computed results for subsequent operations.

** Initial values:
z = 000 . . . 001, one and only one bit in the rightmost position.
q_V = V, the beginning address of the list V.

(38) Read address k of first responder into 1604-B address H.

(39) Set index $q_k = (H) = k$, the address of the implicit variable.

(40) Jump to step (49) on index high $[q_k \geq (0 + q_v)]$. This step tests the address of the next implicit variable with the address of the next variable, implicit or explicit.

(41) An implicit variable has not been found yet. Increment index q_v by 1.

(42) Transfer index q_v to q_k (LDI, q_k, $q_2 = q_v$, h = 0).

(43) Jump to step (45) on index high $[q_k \geq (0 + q_p)]$.** This step tests the address of the next (implicit or explicit) variable against the address of the next operation.

(44) The next variable then belongs to the current operation. Jump unconditionally to step (39).

(45) The current operation has been completed. Random unload into table PDL two words: the address of the operation and the rank accumulator. These are found beginning in the AM address given in 1604-B core position $(q_p - 1)$.

(46) Increment the PDL index by two (initially set to zero).

(47) Increment index q_p by one to adjust the current and next operation.

(48) Jump unconditionally to step (39).

(49) An implicit variable has been found. Random load the next to last element placed into table PDL into the AM location defined by the address $q_k + 4$. The element placed into this location is the AM address of the operation which generates the input. This is the WS_t index.

(50) Random load the last element placed in PDL, shifted one position to the left into the AM position defined by the contents of the 1604-B address $q_p - 1$ plus one. (If the address is $\alpha$and $(\alpha) = 5$, then the desired address is 6.) This is the rank accumulation.

(51) Decrement the PDL index by two.

(52) Perform step (45) and step (46)

(53) Erase the first responder.

---

*Initial values:

$Z = 000 \ldots 001$, one and only one bit in the rightmost position.

$q_v = v$, the beginning address of the list V.

**Initially, q_p is set to $(P + 1)$ where P is the starting address of table P.

(54) Jump unconditionally to step (37).

(55) Continue.

At this time, the command list has been "ranged" and implicit variables have been assigned their generating operation. The following remains to be done:

- Cross reference explicit inputs (items (3)(c) and (3)(e), Paragraph 7.3.3.3).
- Compute norms.
- Sort list into the best processing order. The first task follows:

(56) Reset and search variables.

(57) Advance buffer and search for A values that are neither (¢) nor (zeros).

(58) Read out responders into a list VA, with elements VA_i ($1 \leq i \leq I$), elminating duplications. Read out the count of responders as I.

(59) Set i = 1 as an index.

(60) Search for equality on comparand VA_i.

(61) Write the same unique tag in AM positions, three places after the responders.

(62) Advance buffers. Write a plus (+) in all responders except the last. Write an (*) in this responder.

(63) Using jump on index i and limit I, go to step (60); otherwise, continue.

The norms of the operations require numeric calculation; therefore, some 1604-B processing is in order. There are four types of norm operations: maximum, minimum, sum of, and difference between the variable norms. The variable norms are stored in the second position after the variable indicator. The operation norm is stored in the second position after the operation indicator. The first variable occurs four positions after the operation code and every subsequent variable of the operation begins six positions after the preceding variable.

For the first time 1604-B processing is required. These operations are noted with an asterisk (*) on the numbered step. A relationship between the two lists, one in AM and the other in 1604-B, exists by using the 1604-B beginning address to be modified by an index register which is loaded from a memory cell in which the address of a responder is stored.

(64) Set j = 1.

(65) Read out jth operation and all elements into 1604-B. Let the AM address be A.

(66) Using norm of the variables, compute operation norm.

(67) Search for equality on A.

(68) Write the norm two places after the first responder and two places before every following responder.

(69) If $j \leq R$, the total number of operations, transfer to Step (65); otherwise, continue

This completes the pre-processor algorithm.

The ordering of the operations will be considered in the Run Routine by sorting on (¢) within range. This delays retrieving these variables from the data.

7.3.4.3 Pre-Processor Flowchart. The given flowcharts (Figure 7-4) assume less than the preceding algorithm and show some of the translator's functions.

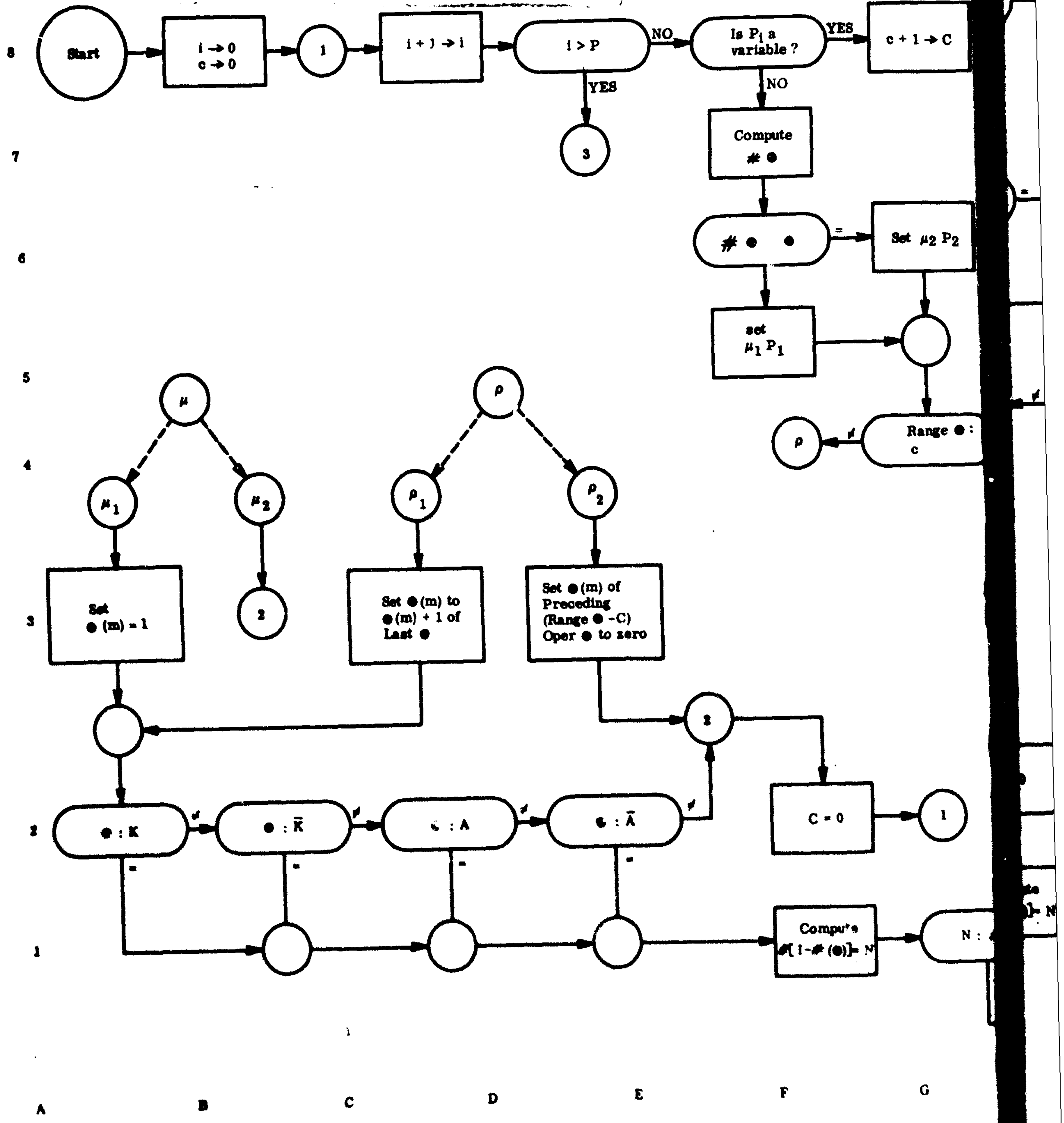
(1) Query Pre-Processor Flowchart (Figure 7-4, Sheet 1). The ppff is scanned initially right to left, the natural processing order. The following is accomplished in this scan.

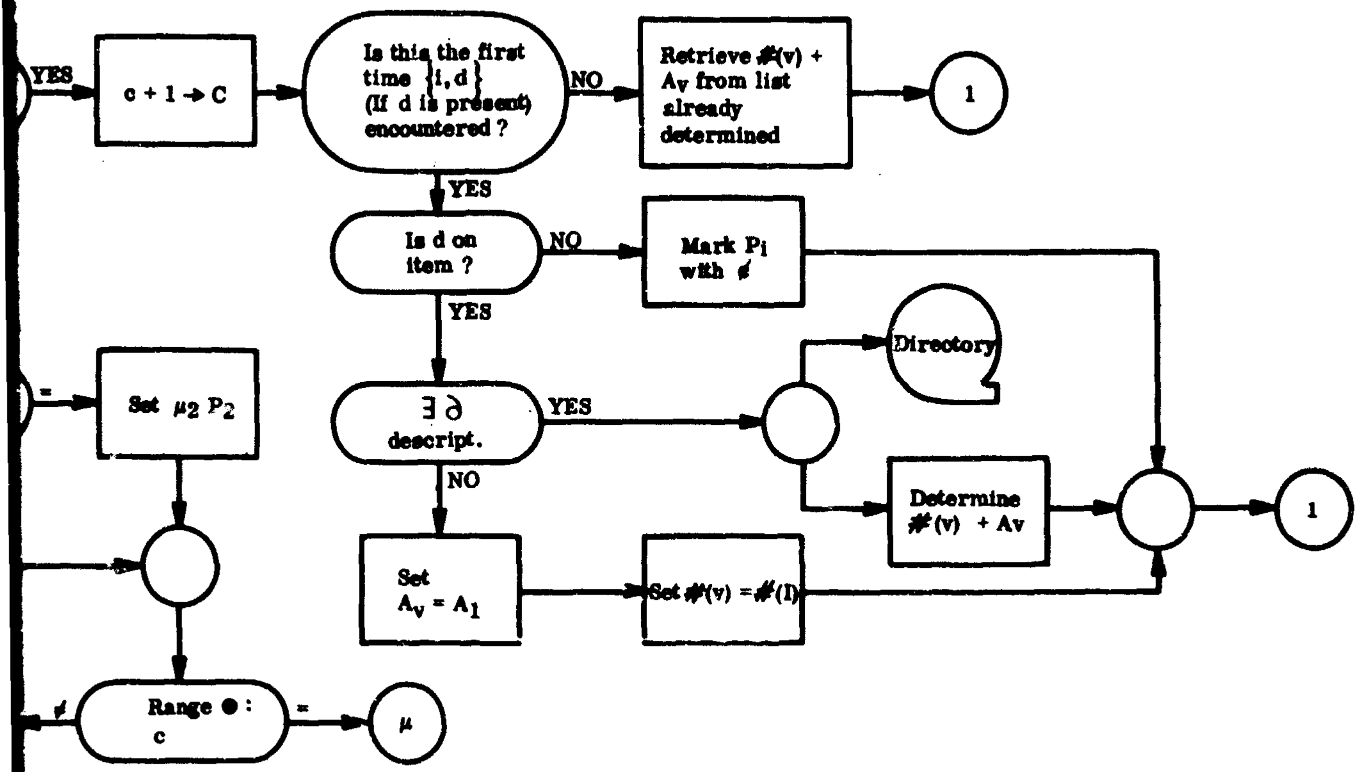
(a) The norm of all variables which are items (appear in the D-Directory) are determined and the address of the logical name list in the directory. This implies that some variables do not appear in ppff. These are:

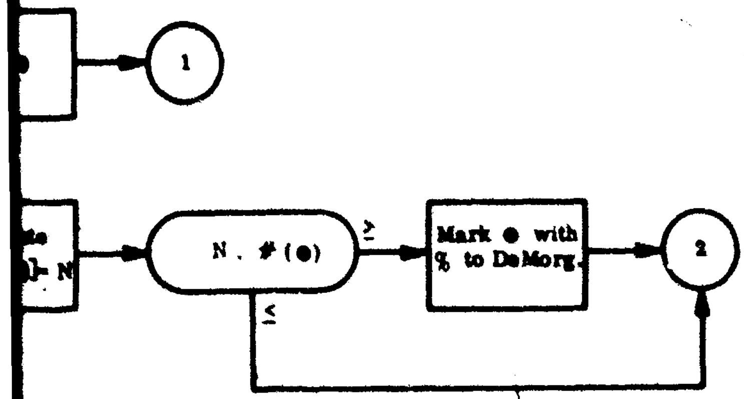
(i) An attribute with no descriptor implies that all records which have that attribute may be queried. Then, the address set is the universe for that variable and the norm is the corresponding number of records.

(ii) A descriptor exists but is not a logical name (ship's superstructure). Then, the list of logical names required is that list of all records which contain the descriptor. The descriptor is marked with (¢). (J6). The norm will be assumed to be the total number of records existing for the type containing that descriptor.

PREVIOUS PAGE WAS BLANK, THEREFORE WAS NOT FILMED







Notes

Scan Polish prefix right to left.
Assume Polish prefix notation of query
Assume range of each operation known.
Assume D-Directory

Let: ●, denote an operation
v, a variable
(v) and # (●), norms of v and ●, number of elements in variable or result of operation
p_i, each element of a query ($1 \le i \le P$).
R to L order

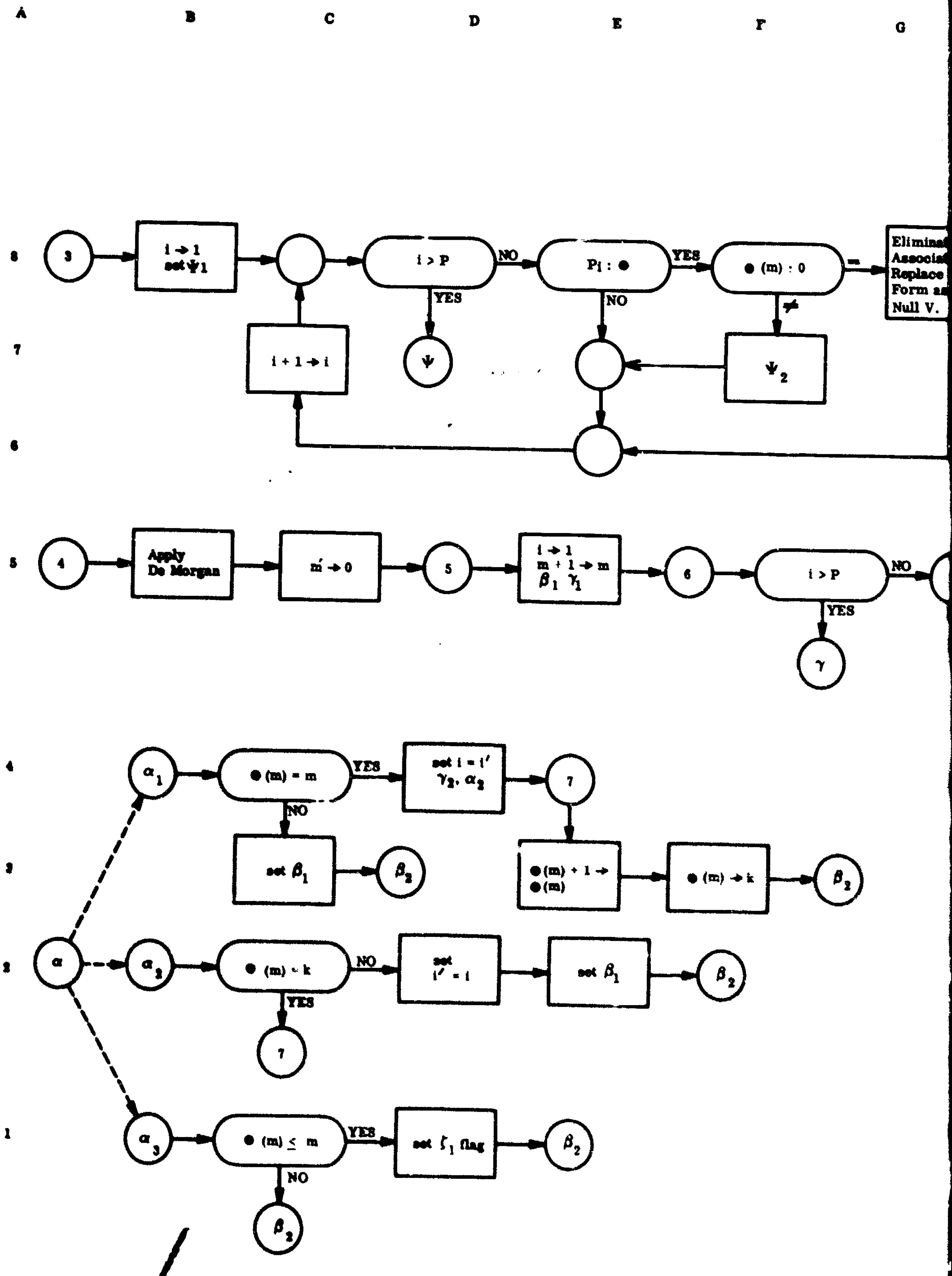
∉, a mark on v to indicate data required from data base
A_v, address of index list for v
I, the universe of all records
I_v, the universe of all records for a particular variable

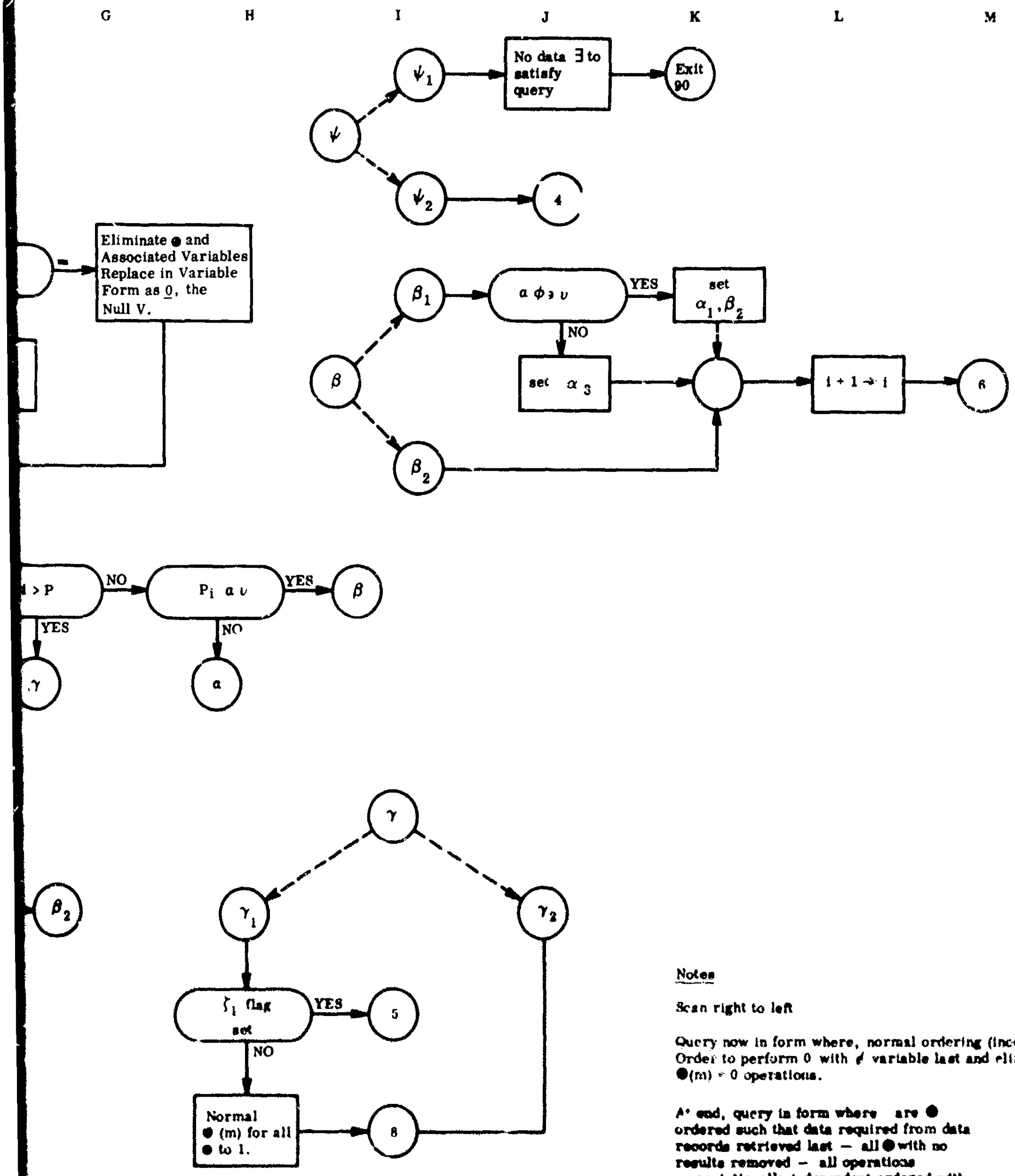
{a, d,} attribute, descriptor set.
m = order of operation ● - ● (m)
c, a counter
%, a mark to indicate DeMorgan-
K, A, are ● for V, and ∩. $\bar{K}$ and $\bar{A}$ are comp.

G H I J K L M

Figure 7-4. Query Pre-Processor
(Sheet 1 of 3)

2


A
B
C
D
E
F
G
8
7
6
5
4
3
2
1
3
i → 1
set Ψ1
i > P
NO
YES
Ψ
Pi : ●
YES
NO
● (m) : 0
=
≠
Elimina
Associa
Replace
Form a
Null V.
i + 1 → i
Ψ2
4
Apply
De Morgan
m′ → 0
5
i → 1
m + 1 → m
β1 γ1
6
i > P
NO
YES
γ
α1
● (m) = m
YES
NO
set i = i′
γ2, α2
7
set β1
β2
●(m) + 1 →
●(m)
● (m) → k
β2
α
α2
● (m) ≤ k
NO
YES
set
i′ = i
set β1
β2
7
α3
● (m) ≤ m
YES
NO
set ζ1 flag
β2
β2



Notes

Scan right to left

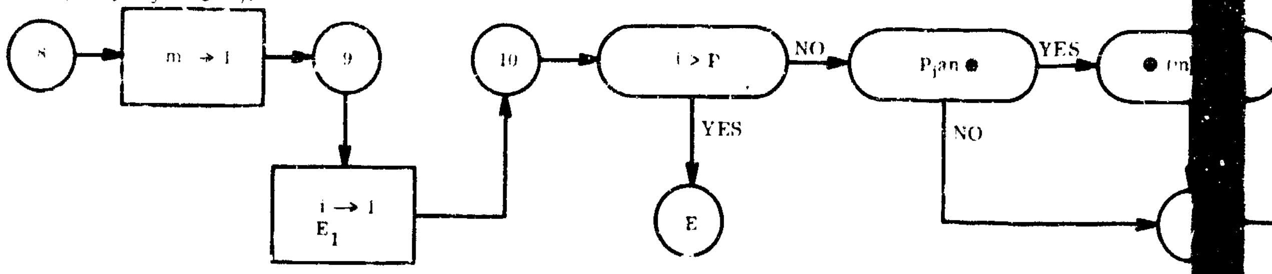
Query now in form where, normal ordering (incomplete) shown. Order to perform 0 with ∉ variable last and eliminate ●(m) = 0 operations.

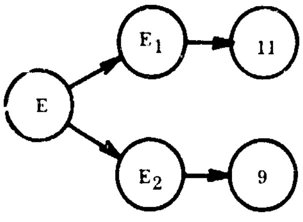
A' end, query in form where are ● ordered such that data required from data records retrieved last — all ● with no results removed — all operations computationally independent ordered with equal precedence.

Figure 7-4. Query Pre-Processor (Sheet 2 of 3)

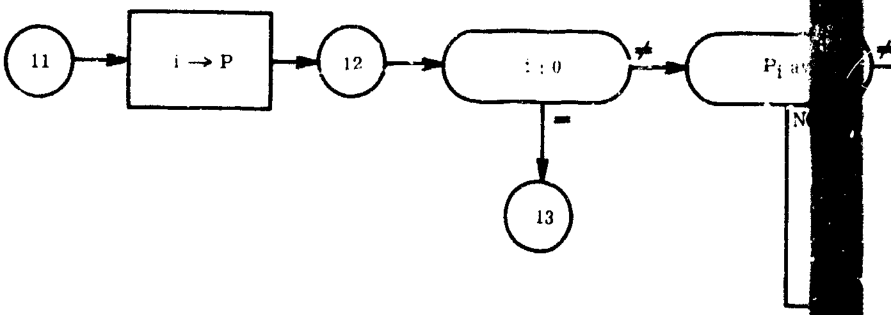
2

8 Assume ● are numbered (1 ≤ n ≤ N) in Query Statement.
Sort ● in Query on ●(n).

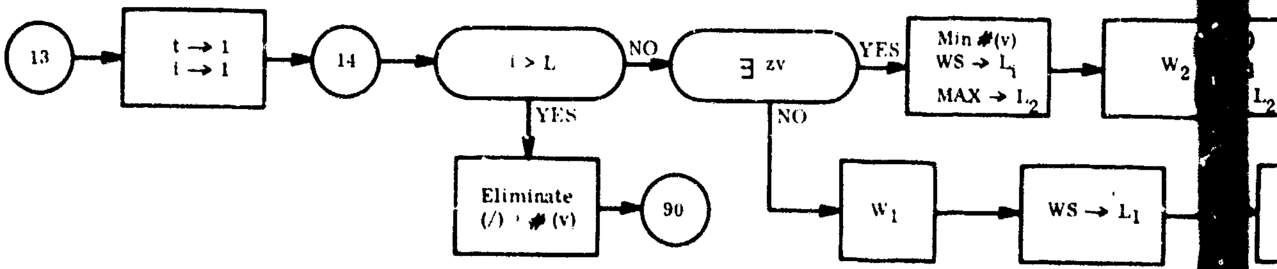




Input List λ, (A Reordered Query List — Scan L to R)



4 Input List λ, Reordered with variables marked (+) or (*)
Memory Assignment



2

1 **Notes:**

Scan right to left.
Let working storage be L_1 and L_2 (see 1231-TN-2).
Let temporary storage be WS_t ($1 \leq t \leq T$; $\sum_t WS_t$ is total storage available, but computed to be total required.
* indicates v will no longer be required in Query when once used.
+ indicates v will be required at a later time.

0A B C D E F G G

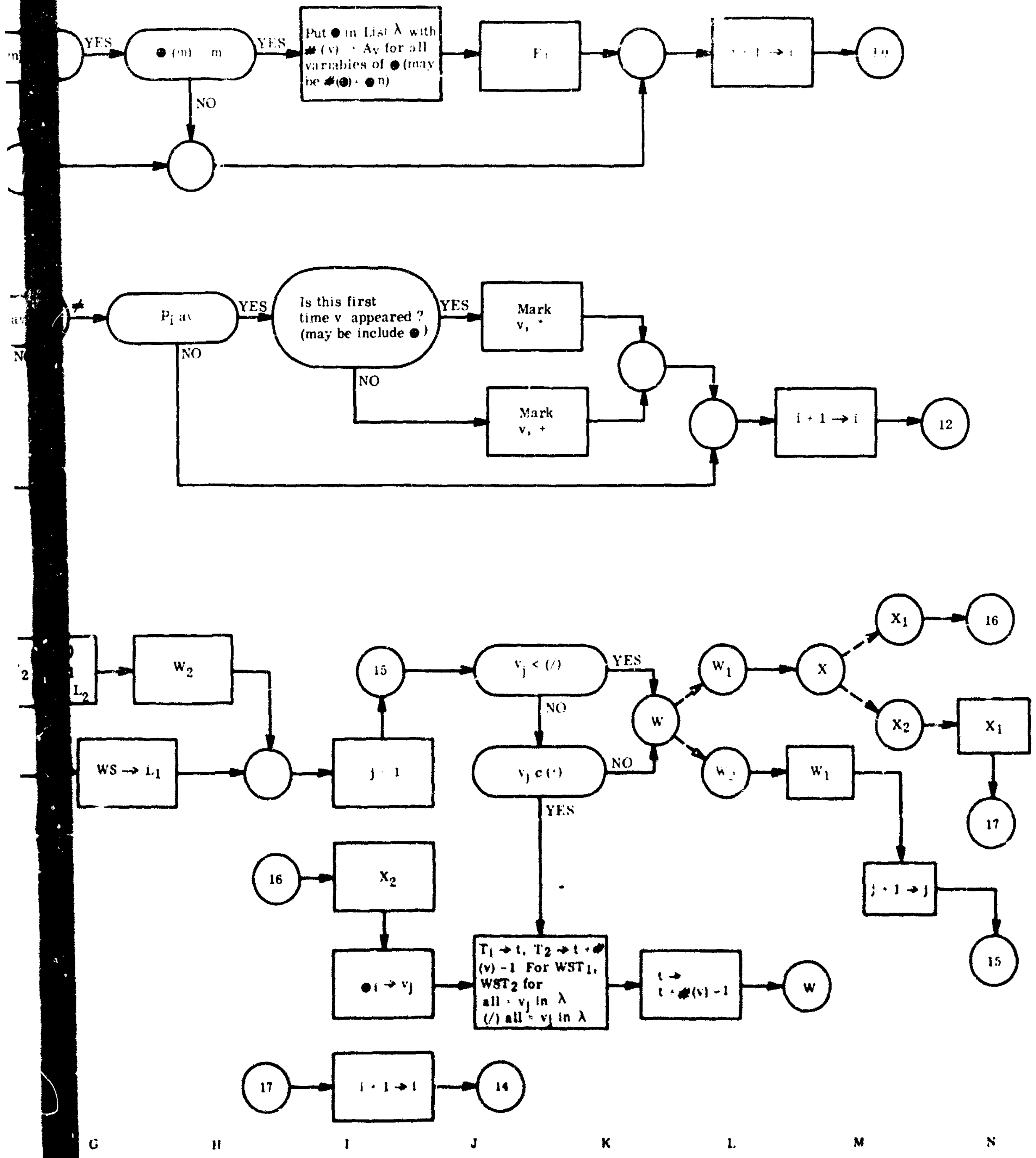

YES
● (m) m
YES
Put ● in List λ with # (v) · Av for all variables of ● (may be #(●) · ● n)
NO
F₁
i + 1 → i
10
≠
Pᵢ a\
YES
Is this first time v appeared ? (may be include ●)
YES
Mark v, *
NO
NO
Mark v, +
i + 1 → i
12
L₂
W₂
WS → L₁
j · 1
15
vⱼ < (/)
YES
NO
vⱼ c (·)
NO
YES
W
W₁
W₂
W₁
X
X₁
16
X₂
X₁
17
j + 1 → j
15
16
X₂
●ᵢ → vⱼ
T₁ → t, T₂ → t + # (v) - 1 For WST₁, WST₂ for all · vⱼ in λ (/) all · vⱼ in λ
t → t + #(v) - 1
W
17
i + 1 → i
14
G
H
I
J
K
L
M
N

Figure 7-4. Query Pre-Processor
(Sheet 3 of 3)

(b) The norm of each operation is computed and the range of the operation checked and the order computed.

(c) Operations that reduce the storage requirements, if they are "De Morganed," are so marked (%).

(2) Query Pre-Processor Flowchart (Figure 7-4 sheets 2 and 3).

(a) All operations which yield no result are eliminated if their norms are equal to zero. If all operations are eliminated, then no data exists to satisfy the query and the pre-processor is complete - as is the query.

(b) De Morgan is applied to ppff and the operations are ordered such that operations requiring data about variables from data records will not be performed until as late as possible.

The rule for doing this operation is: For all operations with a range of m (initially set to 1), increase the range of these operations with at least one or more variables marked with (¢) and provided that there exists at least one operation without a (¢) variable and a range of m. If such is the case, increase m and repeat the rule; if such is not the case exit without increasing the range of any operation further.

(c) Now it is possible to sort the operations in the query ppff into a command list, and list each variable explicitly. For this reason the operations (which result in variables for subsequent operations) are assumed to be numbered. All operations with range (Θ(m)) initially equal to 1 are placed in list λ. (If a variable has a (¢) marked, its range has been increased: therefore, it won't initially appear.) The range is increased until all operations are in the λ list. This list is examined for further refinement.

(d) Each variable, including operation results, is examined to see whether or not it is required for further computation. Thus, if a particular operation is being performed and requires two variables, one of which is marked with a (*) and the other with a (+), it will be known that the area used for storage of the (*) marked variable is now free: whereas, the (+) marked variable must be saved for further computation. This variable marking is accomplished.

(e) It is now known when each variable is required, when it is no longer required, and the approximate size or storage requirements. There are two types of storage areas: working storage and temporary storage. Working storage has been defined in TN-2 as lists L_1 and L_2. Then, for each operation, the working storage assignment is made.

(f) The temporary storage assignment is made. This assignment is a list of "names" for each variable. The number of "names" is equal to the norm of the variable. Each "name" corresponds to one 32-word section of the 1604-B memory. The names for one variable increase by unity and imply contiguity. If a variable is required in more than one operation, the same list of names will be given. The total number of names or numbers (each associated with one 32-word segment), is equal to the sum of the norms of all unique variables. These names (or numbers) are not assigned memory space. (They could be if infinite memory space were available.) This name assignment is accomplished.

7.3.5 General Comments

After developing the associative memory algorithm and the flowchart for the conventional computer (both for the query pre-processor), it is possible to make some observations on each technique regarding such things as computational time, storage space requirements, ease of programming, etc. It is thereby possible to compare both methods of problem analysis.

Processing time is a function of many factors. Among these are the number of operations (steps (1), (17), (30), etc., of the algorithm, and locations G7, F7; etc., of the flowchart); variables (steps G7, F7, etc., of the flowchart); variables (steps (21), (34), etc., and locations F7; G5; etc.); highest ranged operation; etc. It is therefore most difficult to determine an actual processing time without making many assumptions. On the other hand, it is not necessary to determine these times explicitly since the purpose of this paragraph is to compare the two techniques. What is required are the relative time differences between them. This may be developed by considering those functions performed in the associative memory algorithm which are not performed in the conventional memory flowchart and conversely.

Programming difficulty or ease may be compared in the two solution methods by considering how similar functions are performed in both methods. This also gives some timing data.

In the Goodyear associative memory, additional computation time is required to transfer data between the associative memory and the central processor of the 1604-B (of course, this additional time requirement may not be important if subsequent associative memory processing time is appreciably less than conventional processing time for the same function). In the presented algorithm, the entire query in canonical ppff is transferred from the associative memory (step (4) of the algorithm) into the 1604-B and then back into the associative memory in λ canonical form (step (16)). All subsequent operations are performed in the associative memory with the exception of computing norms. In this case, the entire query (an operation at a time) is transferred from the associative memory to the core memory (step (65). Additional data (responder's contents, addresses, etc.) is transferred out of and/or into the associative memory throughout the algorithm, but this time requirement is of the same magnitude as the time required to form the same lists in the conventional memory solution, if such lists are required.

The majority of the operations performed by the query pre-processor are data-dependent operations; that is, the processing performed at a particular step is dependent upon some condition of data determined in a previous step. Some of these conditions are:

(1) If an operation exists.

(2) If the next variable is an implicit variable.

(3) An operation's address (or contents) which precedes (or succeeds) a particular responder.

(4) The class of the next variable (implicit or explicit).

(5) The value of the rank.

The associative memory instructions useful for determining such conditions are the search instructions coupled with I/O instructions. However, a condition usually implies that one of several available processing paths will be subsequently chosen. Then, it is imperative that the ability to determine such conditions exist. This ability exists in the Goodyear associative memory, but is more unwieldly to use in some cases. For example, it is possible to determine whether A is greater than B by use of a search instruction followed by a responder test, but such search instruction destroys previous buffer settings. In other words, the associative memory cannot provide for "nested" conditionals without destroying results of the immediately preceding conditional on a word basis. This nested conditional capability does exist on less than a word basis by use of the complex search instruction. In this case, the intermediate results of a complex search are lost but may be easily formed with a new test. In both the word search schemes and the complex word search instruction, a conditional test must always start at the beginning of the sequence to be tested. This weakness and the resulting programming difficulty can be removed if intermediate buffers can be stored as in the A2 associative memory (Note the implied decrease in time as well.)

The only way to alter a processing path based on a conditional relationship between A and B is to jump on no responders and jump on index high (or low). The preceding paragraph discussed to some extent how the first of these instructions is used. By definition, it is clear how the second set may be used. But how does one retain the current results of a conditional test and proceed to effect another conditional? For example, jump to location C, if the first responder A is greater than the second responder B (while maintaining all responders). The jump on index instruction may be used if A and B are less than 15 bits long; otherwise, it is impossible to perform the desired function. If the data can be related to the addresses then the jump on index instructions may be used. But to use this instruction requires transfer of addresses to the 1604-B, load index, and test instructions. (Consider steps (40), (43), etc.)

The difficulties found in programming which give an increase in computation time are:

(1) It is impossible to load an associative memory register or word from another associative memory register or word.

(2) It is impossible to test the contents of two associative memory words without destroying previous results; thus, conditional testings must be effected in a serial fashion and repeated for each condition.

In using the associative memory for an application, analysis on conditional statements must be performed to reduce nested "If" statements to serial "If" statements whenever possible. If this cannot be accomplished, since intermediate results can not be stored, the entire sequence of nested tests must be performed repeatedly.

Storage comparisons are not too meaningful since the addition of the associative memory increases that hybrid system by some factor. However, instruction storage space is increased in serial processing because of the LDR instruction and the lack of intra-associative memory communication.

Since the A2 (and the A3) can store results of past conditional tests, it appears that the A2 is somewhat "better" (as far as programming ease is concerned) than the Goodyear associative memory. Additionally, since the A3 possesses intra-associative memory communication, it is also better (in the same sense) than the Goodyear Associative Processor.

7.4 RUN ROUTINE

The Run Routine requires as input the command list, λ (Paragraph 7.3.3.3) which may be thought of as a sequence of source coding or instructions to be performed. However, as noted in Paragraph 7.2, each element λ_i of the command list is "performed" by the Run Routine in an interpretive fashion. The Run Routine is controlled by an executive type routine called Controller (Paragraph 7.5). The main function of the Controller is to control input-output operations relative to query processing. The Controller will examine all elements of the command list to determine what operations (depending upon the operation's ranges) may be performed and will control the input of the required data into core memory. The Controller then sets those data elements (B of Paragraph 7.3.3.3) of all such command list entries to λ indicating that the operation may be performed; that is, all inputs required by the operation θ_i are in core memory. Processing control is then transferred to the Run Routine.

The Run Routine then performs only those elements of the command list which have been set by the Controller ($B = \alpha$, Paragraph 7.3.3.3). The particular operation to be performed may be considered as a routine, where θ_i is the beginning address of a sequence of instructions required to effect the operation. Other entries in the related element of the command list contain necessary input parameters such as:

(1) Where the input data is stored.

(2) Where to store the output.

(3) If the variable function is L_1 or L_2* in the related routine.

Then, an algorithm for the run routine follows:

(1) Set $i = 1$.

(2) If $B_i = \alpha$, transfer λ_i to a standard location; otherwise, go to step (4).

(3) Transfer control with a return (to step (4))jump to the related subroutine.

(4) If $i < R$, the total number of operations, increase i by 1 and jump to step (2); otherwise, exit to the output routine.

* L_1 and L_2 are assumed to be two inputs. If θ_i is an operation, then it is sometimes important that a variable be correctly labeled since $\theta_i (L_1, L_2)$ is not necessarily the same as $\theta_i (L_2, L_1)$.

Thus, the Run Routine may be considered to be nothing but a sequence of return jumps to related subroutines. For this reason, the Run Routine need not be considered further.

7.4.1 Subroutine and Operations

Implicit in the preceding discussion of the Run Routine is the tenet that every operation is representable by a subroutine; or as an equivalent, every query may be represented as the final output of a series of subroutines.

Every query is a request for information about one or more items. The requested information may require a "yes - no" answer, a count of the items satisfying the request, a set of descriptors of the requested item(s), or some function such as distance, which uses descriptors of the requested item(s). Additionally, every query contains some conditional statement that must be met for all responding items. This conditional statement involves attributes of the item that must satisfy the given conditions which may also include some function which uses descriptors as arguments. An examination of the queries (Paragraph 3.5) given earlier results in a set of functional operations and a set of conditional operations. The functional operations are:

(1) "Yes or No" depending whether or not one (or more) items satisfy the condition (responds).

(2) "Count" the number of items which respond (responders).

(3) "List" the requested descriptors of the responders.

(4) Using specified (either constant and/or those given by the responders) descriptors, form the following functional operations:

(a) distance between points,

(b) time to traverse the distance where speed is specified,

(c) course to (bearing from) a specified location,

(d) distribution,

and list the results with the responders.

The conditional part of a query involves relationships between descriptors; for example, "List all ships with an aircraft and a doctor aboard" involves the conjunction of the descriptors aircraft and doctor. Then, the conditional operations are:

(1) **Using specified (either explictly given and/ or those given by responders) descriptors for the following functional operations:**

(a) Distance between points.

(b) Time to traverse the distance where speed is specified.

(c) Course to (bearing from) a specified location.

(d) Relationship between specified descriptors:

(i) Equality

(ii) Conjunction

(iii) Disjunction

(iv) Greater than

(v) Less than

(vi) Greater than or equal

(vii) Less than or equal

(viii) Next greater

(ix) Next lesser

(x) Minimum

(xi) Maximum.

(e) Between two specified descriptors

(f) Course change

(g) Course extension

Each of these operations given may be considered to require a subroutine. The input to each subroutine is, therefore, a list (or lists) of descriptors: the output is either a list (or lists) of descriptors or a list (or lists) which show some functional combination of the input descriptors. The listed operations are the genesis for the following routines:

RED — To select logical names of records which satisfy a requested attribute-descriptor pair.

- AND — To find the conjunction of two lists of logical names.
- OR — To find the logical disjunction of two lists of logical names.
- MAX — To find the maximum value of an attribute.
- MIN — To find the minimum value of an attribute.
- NGT — To find the value of a descriptor next greater than a given value.
- NLT — To find the value of a descriptor next less than a given value.
- EQU — To select related logical names of particular logical names from a data file directory.
- DIST — To compute the nautical distance between two points given in latitude-longitude coordinates.
- TIME — To compute the time required to traverse the distance between two points, given the speed of advance.

A description and the coding for each of these routines are given in Appendix C.

Each of these routines assumes that the input data is contained in core memory and that the parameters required have been specified by the entry in the command list. Additionally, each routine is written such that it leaves its output in core memory. Thus, input-output operations are performed, as in most systems, by an executive routine. Then, before timing considerations can be made, input and output functions must be made. In addition, each routine assumes that its input (and output) lists may fit into the associative memory. However, such may not be the case: that is, an input list may be greater in length than the associative memory storage area. In this case, the associative memory dispatcher (which is controlled by the controller) is used.

7.4.1.1 Associative Memory Dispatcher: The purpose of the dispatcher is to handle the following problems:

(1) If the operation is performed on two lists, which list should be loaded into the associative memory?

(2) If the list destined for the associative memory exceeds available storage, what segmentation procedure should be followed to control data transfers and guarantee results?

The context in which these problems are reviewed is as follows. After the Run Routine has determined that the data required for a particular process is in core, it transfers control to a functional subroutine. This subroutine may or may not use the associative memory. If it does use the associative memory, there arise certain housekeeping functions that must be done. The associative memory dispatcher is provided to examine the availability of associative memory storage and make decisions as to which list should be loaded into it, how to segment it if it exceeds the memory capacity, and how to control output lists appropriately. The general nature of this routine will be described in the following paragraphs. (It is assumed that the actual loading of AM from 1604-B core will be controlled by the functional routine, which will be given data to guide this process).

The inputs required by the associative memory dispatcher are:

(1) The data furnished by the Run Routine.

(2) A list controller word (LCW).

The data furnished by the Run Routine is:

(1) The 1604-B core addresses and lengths of input lists, L_1 and L_2.

(2) The 1604-B core address of the input.

(3) The 1604-B core address of the LCW.

(4) The number of available cells in the associative memory.

The LCW is to be associated with a routine. This word will permit the associative memory dispatcher to observe rules of correspondence and linkage required by a particular functional routine. At the time that a routine is specified, the designer or programmer will consider the factors coded into the LCW, and form the LCW vector for that particular routine. When a functional routine is called by the Run Routine, its LCW is made available to the associative memory dispatcher. An example of an LCW control word follows:

Bit Position	Value	Meaning
0	1	Single list operation
1	1	Uniform sequence, L_1, L_2 (one-to-one)
2	1	All L_1 against all L_2
3	1	Simple concatenation of output
4	1	Single element output
5	1	Recursion of output ($L_t \rightarrow L_i$)
6	1	Notify functional routine if segmenting
7	1	Override AM dispatcher decision
8	1	Put L_1 into AM inspite of length
9	1	Put L_2 into AM in spite of length
10	1	Get segment max. from functional routine
.		
.		Reserved for further definition
.		

After computation, the Associative Memory dispatcher provides the following data to the functional routine. (Control is then transferred to the functional routine.)

(1) Identify (L_1 or L_2), core address, and length of input and output lists.

(2) Indication that the current run of the functional routine is (or is not) the last run; that is, more data sublists exist.

After each pass of the functional routine, control is transferred back to the dispatcher (if indicated) or to the next functional subroutine. If control is transferred back to the dispatcher, the purpose is to retrieve more data; that is, more sublists exist to be processed. Then, the functional routine furnishes another LCW. This transfer of control between the functional routine and the dispatcher will result (after a finite number of runs of the functional routine) in the desired final output.

The philosophy of the dispatcher is based on the assumption that computations on successive segments of a list will yield the same results as computations performed on the complete list.

This specification of an associative memory dispatcher and its interlacing logic shows the simplicity with which segmentation can be treated as a system function. The internal logic of the dispatcher may be developed into as sophisticated a decision model as the system requires. By fixing the interface logic at an early system design stage, different allocation policies can be evaluated by manipulating the dispatcher.

The internal logic of the dispatcher is based upon the following rules:

(1) If the operation involves only a single list, and it is an AM operation, load that list into the AM.

(2) If the operation involves two lists and

(a) L_1 is greater than L_2 and

(b) L_1 fits into the AM without segmentation,

load L_1 into the AM.

(3) If the operation involves two lists ana

(a) L_1 is greater than L_2 and

(b) L_1 does not fit into the AM, and

(c) L_2 does fit without segmentation, load L_2 into the AM.

(4) If the operation involves two lists and

(a) Neither L_1 nor L_2 fits without segmentation

(b) L_1 is greater than L_2, load segments of L_1 into the AM.

(5) If LCW contains an override bit on, ignore all the above logic and follow directions of LCW.

This associative memory dispatcher will not be considered further. It is documented in this paragraph to illustrate its need and to indicate an approach to meet this need. For the purposes of this study, it is sufficient to assume that the complete input lists of all functional subroutines may be stored in its entirety within the associated memory.

7.5 CONTROLLER

As indicated in previous paragraphs, the function of the controller is to:

(1) Assign 1604-B core memory locations to input data lists.

(2) Coordinate intermediate outputs (of operations) with succeeding operations (as inputs).

(3) Determine whether data currently in core is needed for further computations.

(4) Coordinate the Run Routine and the input-output functions.

The Controller, in reality, is the first level routine in the query processing. It determines the processing flow within and between queries. After a query has been pre-processed, it is stored in the associative memory in the λ canonical form. The Controller then assumes processing control.

7.5.1 Inputs and Outputs

The main purpose of the Controller is to control the temporary (1604-B) core storage and to interleave the Run Routine and the input functions. In this respect, it has two inputs and one output. The inputs are the command list λ and a map, M, of the core storage. The output is a list called Q whose elements have two entries each: the desired disc address and the allocated core memory location. This Q list is the input to the Input Routine which reorders the elements of Q such that the sequential disc addresses result in a minimal disc running time. When the Q list is exhausted by the Input Routine, control is transferred back to the Controller and then to the Run Routine. This procedure is repeated until all operations of the command list have been perfoı med (all $B_i = \gamma$).

7.5.1.1 The Input Map, M. The input map contains M elements, where each element contains three entries (X_m, Y_m, Z_m), and each entry is associated with one 32-word block of core memory. Thus, the total storage space controlled by the controller is 32M 1604-B core words. For a particular element m of the map M,

X_m = ;, if the associated storage area (1604-B core memory) is free
= +, if the area is not free

= *, if the area will be free after the Run Routine has been performed.

Y_m = WS_t, the tag assigned by the pre-processor (steps (48) and (61) of the pre-processor algorithm, Paragraph 7.3.4.2)

Z_m = the beginning 1604-B core address of the associated block.

7.5.2 Processing Functions of the Controller

The functions of the Controller are:

(1) Makes available working storage areas no longer required and adjusts the command list λ.

(2) Forms the new Q list if additional operations in λ must be performed.

With respect to storage in the associative memory, assume the command list is in the associative memory in λ canonical form (as output from the pre-processor). Also assume that the map, M, is the associative memory. Let the beginning and ending associative memory addresses of the command list and the map be B (λ) and E (λ) and B (M) and E(M), respectively.

The two functions of the Controller are performed as described in the subsequent paragraphs.

7.5.2.1 Free Areas and Adjust Command List. When the Controller is called upon, two situations may exist: Either no routine has been performed (All $B_i = B_i$ for all command list entries), or some routines in the command list have been performed (Some $B_i = \gamma$ or α). For those operations whose $B_i = \alpha$(the associated operation was just performed in the last pass of the Run Routine), the corresponding temporary storage areas must be freed, if possible,provided that the variable is marked with an asterisk in the command list.

(1) For those operations in the command list marked with B = α with variables marked with an asterisk, M = *, between B(λ) and E(λ), from the list of elements WS_t ($1 \leq t \leq T$).

(2) For each WS_t between locations B(M) and E(M) (in location Ym), replace X_m with a semicolon (;).

The associative memory operations for these two steps are not easy to do because they both involve searching between address sets and then checking for contents of an address which is a computable number of places after the responder. For this reason, these operations have been replaced by:

(1) For those operations in the command list marked with B = α, between locations B(λ) and E(λ), set B = γ.

(2) For those X_m in M (between B(M) and E(M))equal to an asterisk (*), set X_m equal to a semicolon (;).

7.5.2.2 Form the Q Input List. Now, it is desired to form a list of addresses for the Input Routine and adjust the command list for the next iteration of the Run Routine. The pre-processor determined the range of each operation, but did not reorder the elements of the command list on the range. This reordering may be done implicitly by having the programming assign sequential tags to each successive operation to be performed. Let this tag be incorporated in the B element of the command list: that is, B = α, t, where t = 1, 2, . . . implies that the t^{th} operation to be performed on the next iteration of the Run Routine is that operation with B = αt. The ordering rules are (Paragraph 7. 3. 3. 2):

(1) For those operations which have not been performed (B = β), perform that operation with the lowest range. If all B = γ , exit.

(2) If more than one operation results from step (1), perform those operations which do not require data base (only directory) inputs. This fact is known if A is not equal to (¢) (Paragraph 7. 3. 3. 3).

(3) If more than one operation results from step (2), perform the operations in a first come, first serve order. In this case, ignore those operations with A = ¢.

(4) If all operations in step (2) have A = ¢, perform these operations in a first come, first serve order.

The algorithm to accomplish this ordering (adjusting the command list for the next Run Routine iteration) and preparation of the Q list follows:

(1) Determine those operations which have a B = β . If no responders exist, the query is complete. Exit to pick up the next query.

(2) For those operations with B = β , determine the set with the same minimum range.

(3) Delete those operations which have a variable which requires data file material; that is, A = ¢. If the resulting set is empty, go to step (9); otherwise, continue.

(4) For each successive operation of the resulting set, determine for all variables of that operation:

(a) If A = O, (the data is already in the memory) do nothing.

(b) If A = $\{A\}$, (≠ O, ≠ ¢), continue.

(5) Place WS_t of the variable defined by step (4) in those Y_m whose associated elements of the map, M, are free (X_m :) and are contiguous for N = # (¢) (the variables norm). In other words, "write" WS_t of the variable in those Y_m where:

(a) X_m = ; and

(b) $(Z_m) + 32 = (Z_m + 1)$ for N elements.

Place $\{A_v\}$ and $\{Z_m\}$ defined in the Q list. If it is impossible to determine a sufficient number of contiguous spaces for all variables in the operation, go to step (8); otherwise continue.

(6) Now for each such variable of the current operation, search the command list for similar variables (A $\{A_v\}$) and set these A equal to zero to indicate that the data is already in the memory.

(7) For the current operation, (all A 0) examine the M value. If M is an asterisk (*), place this asterisk in all X_m which have Y_m WS_t.

(8) If more operations of the same range exist, go to step (4). If not, increase the range by one and go to step (2), provided that more unassigned working storages exist (there exist elements in M with X_m :). If not, exit the Input Routine with Q.

Notice that the addition of step (7) above serves no purpose in regard to the task at hand; however, this step permits the simpler processing given in the preceding paragraph.

7.5.3 A Goodyear Associative Processor Algorithm

The algorithms given in the two preceding paragraphs may be translated into either a conventional memory or an associative memory, oriented algorithm. This conversion is oriented to the Goodyear Associative Processor.

Because there exist two lists, the command list, and the map (M) in the associative memory, it is important to point out that each search is relative to a specific section of the memory. Then, the subroutine SPO (Appendix C) which has the following two options will be used extensively.

(1) Search responders.

(2) Search between addresses.

The first part of the Controller (Paragraph 7.5.2.1) is accomplished as follows:

(1) Between B (λ) and E (λ) search for operations.

(2) Advance buffer and search for equality with α in the comparand register.

(3) If no responders, jump to step (12).

(4) Write (γ) into responders.

(5) Between B (M) and E (M), search for X_m.

(6) For responders, search for equality with an asterisk (*) in the comparand.

(7) Jump to step (9) on no responders.

(8) Write a semicolon (;) into responders to free the associated working storage space.

(9) Between B (λ) and E (λ), search for operations.

(10) Advance buffer and search for inequality with (γ) in the comparand.

(11) If no responders, jump to final exit since the query has been processed; otherwise, continue.

(12) The second part of the Controller operates on the command list and the map. Its output is a list Q. This list will be formed in the 1604-B core memory. Set R, a minimum range counter, equal to one. Search for operations.

(13) Advance buffer and equality search for (β). The responders indicate those operations which have not been performed.

(14) For the responders, advance the buffer to locate the range of the operation and equality search for R. The responders are those operations which have a minimum value.

(15) Now, these responding operations may require as input one or more variables. If any one of an operation's variables requires data file material (A = ¢), remove that operation from the list of responders, provided that the resulting list contains at least one operation. This testing is more difficult to do.

Advance all responders a sufficient number of places to determine A of the first variable. Search for equality on ¢. Erase these responders. Advance the remaining responders of those operations which have a norm of at least two, a sufficient number of places to determine A of the second variable. Search for equality on ¢. Erase these responders. Continue this procedure for all values of the operations' norms. (In this application, the highest valued norm is two). The responders, or rather the operations related to the responders, must be assigned temporary working storage.

(16) Jump to step (32) on no responders resulting from the operations of step (15).

(17) An operation will only be performed if working storage is available for all variables of the operation.

For the first variable responder, retrieve the norm of the variable, the working storage tag, and the address set $\{A\}$ for the variables where A ≠ O. These elements are given in the command list as # (v), WS_t, and A, respectively.

(18) Jump on no responders (all variables of the operation have been assigned to storage) to step (25).

(19) For the first "responding" variable of step (17), search the map, M, between B(M) and E(M) for equality with a semicolon in the comparand to identify working storage areas that are free.

(20) Determine whether there exist # (v) contigious storage spacesas follows:

(a) Read out address of first responder = H.

(b) Read out count of responders between associative memory address H and [H + # (v) - 1] .

(c) Using jump on index instruction between count obtained in step (b) and # (v), determine whether spaces exist. If so, jump to step (21): otherwise, continue.

(d) Erase first responder.

(e) Jump on no responder to step (24).

(f) Jump unconditionally to step (a) above.

(21) For the responding set of step (20) (a subset of the responding set of step (19), form a temporary list of responding addresses H_i $[1 \leq i \leq \# \ (v)]$ and a temporary list of Z_m. Pair each Z_m with an address of $\{A\}$.

(22) Erase first variable responder from the command list and responder X_m of step (19).

(23) Jump to step (26) on no variable responders for the current operation; otherwise, jump unconditionally to step (16) to process the "next" variable.

(24) At this point, it is impossible to process the current operation; that is, to assign the required storage for at least one variable in the operation. Since some assignments may have been made (on a temporary basis), reset the responder X_m erased in step (19) for this current operation. In addition, delete all Z_m and $\{A\}$ from temporary storage of step (20).

(25) Erase the first operation responder.

(26) Jump to step (27) on no responders; otherwise, jump unconditionally to step (16) to process the next operation.

(27) The current operation may be performed by the command list. Move the temporary storage of step (20) into permanent storage as part of list Q. Jump unconditionally to step (24) to process the next operation.

(28) The assignment for this operational range value is complete. Search map, M, between B(M) and E(M) for equality with a semicolon in the comparand. The unerased responders were not assigned storage. (It will be assumed that some elements were assigned storage).

(29) Jump to step (31) on no responders.

(30) Increase R by one.

(31) If a range of R exists in the command list (operation search, advance buffer two positions, equality search with R), jump to step (14); otherwise, continue.

(32) The Controller has assigned all possible working storage spaces, jump to the Input Routine with the Q list.

7.5.4 General Comments

Again, as in the pre-processor, the required data manipulation is very data dependent (consider step (15) of the algorithm). The processing requires determining the status of elements related to an operation. At times, the status of one element (a variable) of an operation depends upon the status of subelements (norm, value of A, etc.). The general comments regarding "nested" conditionals given in Paragraph 7.3.5 are also applicable. In addition, note that the testing of these sub-elements (and elements) is accomplished by particular responders when in reality the status desired is for the related higher ordered responder, a variable or an operation. This observation and the difficulties encountered in both the pre-processor and the Controller regarding data dependent processing are greatly alleviated by a technique termed "Tag Memory" described in Paragraph 7.9.

7.6 INPUT ROUTINE

The Input Routine is an executive level routine (and may be considered part of the Controller) with the objective of bringing in data from the disc file in the most expeditious manner. The disc addresses of the data desired and the 1604-B core addresses where such data is to be stored are given in the input list Q. This input Q list, the output of the Controller is reordered by the Input Routine in an order which will result in the minimum disc access times. To understand this ordering, it is essential that the CDC 1619 Magnetic Disc File Controller and the CDC 818 Disc File System be understood. The pertinent items of these units are given in the following paragraph.

7.6.1 CDC 1619 Controller and CDC 818 Disc File*

From the given reference, the pertinent times to accomplish specific disc tasks are:

(1) Switch between files (4 milliseconds)

(2) Switch between discs (23 milliseconds)

* Reference Manual Control Data 1619 Magnetic Disc File Controller, Publication No. 60044900, February, 1964, Control Data Corporation.

(3) Position (Stroke) in milliseconds

- Maximum: (3P + 40)
- Minimum: (2.42P + 22.6)
- Average: (2.66P + 32.33), where P is the number of positions moved (P 63).

(4) Confirm position (41 milliseconds)

(5) Latency

- Maximum (52 milliseconds)
- Average (28 milliseconds)

(6) Switch heads (100 microseconds)

From the above times, it can be seen that one may read from a pre-set position of a new disc in less time on the average (may be equal) than one may read from an adjacent (P=1) position of the same disc. Thus, to minimize times, the strokes should be minimal and the disc should be switched without switching the position. This is nearly impossible in a practical sense.

The physical constraints deal with such things as the number of positions, etc. This may be illustrated by a disc address:

(1) Bits 0-6 specify one of 128_{10} blocks of data which use the addresses 000 through 177_8.

(2) Bits 12-17 specify one of 64_{10} possible positions of the position arm using positions 00 through 77_8.

(3) Bits 18-21 specify one of 16_{10} possible discs using disc numbers 00 through 17_8.

(4) Bits 22 and 23 specify one of four files. Since the assumed system has only one file, these bits will be ignored.

An address specifies one disc out of 16 and one position out of 64 and one block out of 128.

Each disc has eight read-write heads on one arm. Four heads are allocated to the inner track and four heads are allocated to the outer track. There are 20 blocks on an outer track and 12 blocks on an inner track. The addresses of these blocks relative to their tracks are given below:

<table>
<tr><td>1</td><td colspan="2">177</td><td colspan="2">000</td><td colspan="2">001</td><td colspan="2">002</td><td colspan="2">003</td><td colspan="2">004</td><td colspan="2">005</td><td colspan="2">006</td><td colspan="2">007</td><td colspan="2">010</td><td colspan="2">011</td><td colspan="2">012</td></tr>
<tr><td>2</td><td colspan="2">013</td><td colspan="2">014</td><td colspan="2">015</td><td colspan="2">016</td><td colspan="2">017</td><td colspan="2">020</td><td colspan="2">021</td><td colspan="2">022</td><td colspan="2">023</td><td colspan="2">024</td><td colspan="2">025</td><td colspan="2">026</td></tr>
<tr><td>3</td><td colspan="2">027</td><td colspan="2">030</td><td colspan="2">031</td><td colspan="2">032</td><td colspan="2">033</td><td colspan="2">034</td><td colspan="2">035</td><td colspan="2">035</td><td colspan="2">037</td><td colspan="2">040</td><td colspan="2">041</td><td colspan="2">042</td></tr>
<tr><td>4</td><td colspan="2">043</td><td colspan="2">044</td><td colspan="2">045</td><td colspan="2">046</td><td colspan="2">047</td><td colspan="2">050</td><td colspan="2">051</td><td colspan="2">052</td><td colspan="2">053</td><td colspan="2">054</td><td colspan="2">055</td><td colspan="2">056</td></tr>
<tr><td>5</td><td>057</td><td>060</td><td>061</td><td>062</td><td>063</td><td>064</td><td>065</td><td>066</td><td>067</td><td>070</td><td>071</td><td>072</td><td>073</td><td>074</td><td>075</td><td>076</td><td>077</td><td>100</td><td>101</td><td>102</td></tr>
<tr><td>6</td><td>103</td><td>104</td><td>105</td><td>106</td><td>107</td><td>110</td><td>111</td><td>112</td><td>113</td><td>114</td><td>115</td><td>116</td><td>117</td><td>120</td><td>121</td><td>122</td><td>123</td><td>124</td><td>125</td><td>126</td></tr>
<tr><td>7</td><td>127</td><td>130</td><td>131</td><td>132</td><td>133</td><td>134</td><td>135</td><td>136</td><td>137</td><td>140</td><td>141</td><td>142</td><td>143</td><td>144</td><td>145</td><td>146</td><td>147</td><td>150</td><td>151</td><td>152</td></tr>
<tr><td>8</td><td>153</td><td>154</td><td>155</td><td>156</td><td>157</td><td>160</td><td>161</td><td>162</td><td>163</td><td>164</td><td>165</td><td>166</td><td>167</td><td>170</td><td>171</td><td>172</td><td>173</td><td>174</td><td>175</td><td>176</td></tr>
</table>

Notice that the inner block addresses are modulo 12 and the outer modulo 20.

From one disc revolution with one position, it is possible to read blocks from the inner and outer zone (by head switching). For example, the following blocks may be retrieved in one disc revolution (057, 060, 015, 110, 033, 070, . . .). A maximum of 20 blocks may be read in one revolution.

7.6.1.1 Data Base Allocation on Disc. Incidental to the main discussion on the Input Routine, it is possible to make some observations regarding the storage of data base information on the disc file. To do this, certain statistics are required.

(1) Data Base Statistics. The assumed data base statistics are:

- Seven ships for each port and on the average 65 percent of the ships are not in port.
- Every ship may carry cargo and possess weapons. However, 10 to 15 percent of the total number of ships possess different characteristics. Thus, if there are 1,000 ships, there exist 100 to 150 different cargo characteristics among these 1,000 ships.
- There are five personalities for each ship.
- There are a total of 108 descriptors divided in the data files as indicated.
- On the average, 33 percent of all possible attributes are specified in data records.

These statistics generate the following data base size characteristics:

- Dictionary requires 10 blocks (absolute)
- File locator requires five blocks (absolute)
- File directories (relative)

(a)	Static	10 records/	10 words/	record
(b)	Dynamic	6 "	8 "	"
(c)	Port	2 "	10 "	"
(d)	Personality	50 "	3 "	"

(e)	Weapons	1 record/	12 words/	record
(f)	Operational	2 "	12 "	"
(g)	Cargo	1 "	12 "	"
(h)	Propulsion	2 "	12 "	"

(2) Data Base Allocation on Disc. From Paragraph 7.6.1, the following conclusions regarding the storage of data may be made:

(a) Sequentially required data from a file should be stored on as few discs as possible.

(b) Randomly required data may be stored anywhere on the disc.

(c) If data is required from a file always starting at the same physical location, such a file should be on one disc if possible and the disc's read-write positioner should be returned to the initial position.

These conclusions and the statistics of the preceding paragraph along with the following disc allocations are feasible:

Disc	Positions	Contents
1	2-64	Static Ships File (Data)
2	all	"
3	all	"
4	all	"
5	all	"
6	all	Dynamic Ships File (Data)
7	all	"
8	all	"
9	all	Operational Characteristic File
10	all	Propulsion Characteristic File
11	all	D-Directory
12	all	Personality File Directory
13	1-45	Static File Directory
13	46-64	Personality File Directory (continued)
14	1-20	Dynamic File Directory

Disc	Positions	Contents
14	21-64	Cargo Characteristic File
15	1-10	Port File Directory
15	11-64	"
16	1-5	Weapons Characteristic Directory
16	6-15	Operational Characteristic Directory
16	16-23	Cargo Characteristic Directory
16	24-35	Propulsion Characteristic D:rectory
16	36-64	Weapons Characteristic File
1	1	Space Available File

7.6.2 Rules for Reordering the Input List Q

Paragraph 7.6.1 detailed sufficient disc file constraints to permi' the derivation of the following ordering rules (and assumptions):

(1) It is timewise equivalent to reposition on a new disc as it is on the same disc.

(2) If only one address is given in Q for a particular disc, this address is placed into a list I_1.

(3) If more addresses remain in Q, there must exist more than one address required from each disc referenced. Then, if α_D is the D^{th} element of a list which stores the last referenced position of the D^{th} disc, all addresses referencing the D^{th} disc are sorted into increasing or decreasing order depending on the current position (α_D). The rule used assures the minimal travel of the positioner arm.

(4) If for a particular disc, there exists a set of ordered addresses and unique position values (one block required for any one position), this set, along with similar sets for all such discs, is placed in the list I_2.

(5) If Q is not empty, the only thing that remains is the ordering of block addresses/position. The table in Paragraph 7.6.1 indicates the head (1-8) which reads the specified block. This table also indicates relative block positions. The characteristics of the eight read-write heads are such that one may switch between heads in 100 microseconds. This is ample time to identify adjacent* blocks. Thus, the maximum time to read any number of blocks/position is 8 revolutions @ 52 ms = 416 ms.

* If inner zone, block k is adjacent to block r if k = k_1 (mod 12) and r = (k_1 + 1) (mod 12). In the outer zone, block k is adjacent to block r if k = k_1 (mod 20) and r = (k_1 + 1) (mod 20). Between the two zones, an ordering system is needed.

(6) For blocks with the same position for the same disc, the ordering is accomplished by this algorithm:

For all addresses with the same disc and position, but with different block addresses, determine all (mod 20) addresses. From the table in Paragraph 7.6.1 it is seen that these may be divided into four equal subsections composed of five consecutive mod 20 addresses. In like manner, divide the mod 12 addresses.

If:

(1) All of the first two mod 20 addresses are absent, one of the first mod 12 addresses is placed in I_z.

(2) All of the second, third, and fourth mod 20 addresses are absent, one of the second (mod 12) addresses is placed in I_z.

(3) All of the fourth and fifth addresses (mod 20) are absent, one of the third (mod 12) addresses is placed in I_z.

(4) However, if any one of the (mod 20) addresses is present, it is placed in I_Z and the corresponding rule or rules regarding the (mod 12) addresses are not considered.

(5) This procedure is repeated for all four subsections.

(6) Whenever an address is placed in I_Z, it is eliminated from Q, but there may exist four addresses equal to any value mod 12 or mod 20. Thus, at most, eight decision paths are required.

When all discs have been examined (Q is empty), the complete list I_Z is formed.

The reordered list Q is a juxtaposition of the lists I_1, I_2, and I_3. The complete reordered list (I_1 I_2 I_3) may be considered as the output, or each individual list may be considered as output and executed (that is, read the disc address data into the core memory) address while the next list is being generated.

7.6.3 An Associative Memory Algorithm for the Input Routine

Assume the input list Q is in the associative memory in such a form that the disc addresses are listed in the lower memory and the core memory addresses are listed in the upper memory. Let there be mapping (mod 1024) so that the disc address in memory cell m is related to the core address in memory cell (m + 1024).

The contents of a comparand register, when searching disc addresses, may be considered to consist of fields:

- Bits 1-6, block address ($1 \leq B \leq 128$)
- Bits 12-17 P, a position address ($1 \leq P \leq 64$)
- Bits 18-21 D, a disc address ($1 \leq D \leq 16$)

Form I_1.

(1) Set D = 1, and mask out B and P. Set $q_k = 0$.

(2) Search for equality with D in the comparand.

(3) Jump to step (5) on one and only one responder.

(4) Increase D by one if $D < 16$, and jump unconditionally to step (2). If $D \geq 16$, jump unconditionally to step (9).

(5) There is one and only one address which uses disc D; read out the contents of the responder as the address. Erase the responder.

(6) Increase the output index q_k by one.

(7) Read out the contents of the upper half of the memory as the 1604-B address.

(8) Increase q_k by one, and jump to step (4).

(9) The list I_1 has been formed. Either execute I_1 or store it. If I_1 is executed, set $q_k = o$.

(10) Assume the table containing α_D exists. Set D = 1, and mask out B and P.

(11) Search for equality with D in the comparand.

(12) Jump to step (33) on no responders.

(13) Mask out B and D and search for minimum, saving previous buffer. (RS goes to E.)

(14) Read out contents of responder into 1604-B memory cell, min p.

(15) Reset D register. Search for maximum, saving previous buffer.

(16) Read out contents of responder into 1605-B memory cell, max p. Reset D register.

(17) In 1604-B pe:-form:

(a) If $\alpha_D \leq$ min p, jump to step (f).

(b) If $\alpha_D \geq$ max p, jump to step (e).

(c) Then min p $< \alpha_D <$ max p. Compute:

(i) A = (α_D - min p)

(ii) B = (max p - α_D)

(d) If A $\leq$ B, jump to step (f).

(e) The addresses are to be ordered in increasing order. Set storage cell K to min p and set NHC (Next Higher than Comparand) instruction into step (31). Jump to step (18).

(f) The addresses are to be ordered in decreasing order. Set storage cell K to max p and set NLC - (Next Lower than Comparand) instruction into step (31). Continue.

(18) The buffer has been retained in step (15). It exists in the E buffer; therefore, it may be reset when desired. Set D register from E register.

(19) Jump to step (25) on no responders.

(20) Load the first responder into the comparand register. Mask out B and D.

(21) Erase first responder.

(22) Equality search.

(23) Jump to step (18) on no responders.

(24) If there is a responder, then there are at least two addresses which reference the same position of the same disc. Therefore, reordering is required on a block level for this disc. Jump to step (34).

(25) Every disc address for the current disc requires a unique position; therefore, to reset busy bits, write through a mask of zeros in all responders after D is reset.

(26) Load storage cell K into the comparand.

(27) Search for equality.

(28) Read out the contents of the responder and the contents of the address of the responder in the upper half into I_2. Increase output index.

(29) Erase the responder. Reset the D register (from E).

(30) Jump to step (33) on no responders.

(31) Perform either NHC or NLC on responders with P masked.

(32) Load the responder into the comparand register and jump to step (28).

(33) The current disc has been processed. Increase D by one provided that $D < 16$. Set the mask to mask out B and P and jump unconditionally to step (11). If $D = 16$, both lists I_2 and I_3 are complete. If I_1 was executed, then execute I_2 and I_3 juxtaposed; otherwise juxtapose I_1, I_2, and I_3 and execute the combined lists.

(34) The current disc has addresses that require block ordering. Some addresses may not require block ordering, only positional ordering. Return jump, performing steps (25), (26), and (27).

(35) Read count of responders into memory cell H.

(36) Load H into index register q_1.

(37) Jump on index high if q_1 (the number of responders) is greater than or equal to $g = 2$ to step (40); otherwise, continue.

(38) There exists only one responder, return jump, performing steps (28), (29), (30), (31).

(39) Jump unconditionally to step (35).

(40) The current position has more than one block address. These positions are indicated by the current status of the D register. The instruction BLC, Between Limiting Comparands, will be used with the limiting comparands of A and B. Let a table exist in the 1604-B core memory of these comparands. The contents of this table are [012, 000, 027, 013, 043, 030, 057, 044, 103, 060, 127, 104, 153, 130, 177 and 154...]. Let this table be T with elements t_i, $(1 \leq i \leq 16)$. A difference F is also needed. This difference 12i for $i < 8$ and (057 and 20i) for $i \geq 8$. In like manner, an index q_p is needed. For $i \geq 8$, q_p is equal to the address of the last element of Q; for $i \geq 8$, q_p is equal to twice this address. Then all mod 12 addresses are stored between q_p and $2q_p$, and all mod 20 addresses are stored between $2q_p$ and $3q_p$.

(41) Perform BLC; $A = t_i$, $B = t_{i+1}$

(42) Jump to step (48) on no responders.

(43) Read out first responder contents.

(44) Subtract (a 1604-B operation) F from the block position.

(45) Read out the first responders, address = H.

(46) Write the results of step (44) back into the associative memory into an address (H plus q_p = the address of the last element of the list Q).

(47) Erase the first responder and jump to step (42).

(48) Reset the D register. Increase i by 2 if i < 16 and jump to step (41). If i = 15, continue.

(49) At this time, a comparable disc address exists — modified modulo 12 or modulo 20 which is stored in an associative memory cell q_p or $2q_p$ cells higher than the responders of step (40). Set B = 000 and B' = 000. Set masks for D and P.

(50) Load B into comparand. Set $q_p = 2q_p$. Reset D register.

(51) Search for equality between addresses $2q_p$ and $3q_p$.

(52) Jump to step (57) on no responders.

(53) Read out the address H of the first responder.

(54) Erase the first responder.

(55) Read out the contents of location H - q_p from the upper and lower memories into I. Advance the index q_p by two.

(56) Set step (62) to step (67).

(57) Increase B by one and load B into comparand. Reset D register.

(58) Search for equality between addresses $2q_p$ and $3q_p$.

(59) Jump to step (62) on no responder.

(60) Perform steps 54, 55, and 56.

(61) Set step (76) to step (81).

(62) (Initially no operation).

(63) Load B' into the comparand. Set $q_p = q_p$ and reset the D register.

(64) Equality search.

(65) Jump to step (67) on no responders.

(66) Perform steps (54) and (55).

(67) Increase B by two and load B into comparand. Reset D register. Set $q_p = 2q_p$.

(68) Equality search.

(69) Jump to step (71) on no responders.

(70) Perform steps (54), (55), and (61).

(71) Increase B by three and load B into comparand. Reset D register.

(72) Equality search.

(73) Jump to step (76) on no responders.

(74) Perform step (70).

(75) Set step (85) to step (90).

(76) (Initially, no operation.)

(77) Increase B' by one and load B' into comparand. Set $q_p = q_p$.

(78) Equality search.

(79) Jump to step (81) on no responders.

(80) Perform steps (54) and (55).

(81) Increase B by four and load B into comparand. Set $q_p = 2q_p$.

(82) Equality search.

(83) Jump to step (85) on no responders.

(84) Perform steps (54), (55), and (75).

(85) (Initially, no operation)

(86) Increase B' by two and load B' into comparand. Set $q_p = q_p$. Set D register.

(87) Equality search.

(88) Jump to step (90) on no responders.

(89) Perform steps (54) and (55).

(90) Reset steps (62), (76), and (85).

(91) If B $\leq$ 14, jump unconditionally to step (50). Otherwise,contin

(92) Set D register. Jump to step (33) on no responders; otherwis jump unconditionally to step (49).

The input list Q has been completely reordered in the desired manner.

7.6.3.1 General Comments. The preceding algorithm requires the use of sever programming procedures not used explicitly in the other algorithms given in this section. These follow:

(1) To avoid buffer shifts to read out data related to responders ir the lower memory, this data was stored in the same relative location in the upper memory. This device, while not necessa for processing, was introduced at this time to illustrate the T Memory described in Paragraph 7.9.

(2) Loading of comparand with the actual data (which is not know *a priori*) determined as a result of other data conditions.

(3) The use of variable connectors, that is, data content determin paths.

In addition, the difficulties encountered in previous algorithms because o nested conditionals also occurred in this algorithm. To alleviate this difficulty to s degree, the upper memory was not used for data examined to determine these condi to permit saving the results of one conditional in the upper response store (D regist while testing the data for another conditional.

A variable connector is defined as the initial junction point of several sub quent programming paths. The particular path to be taken is determined before the point is reached. A variable connector differs from a branch point in that intermed processing — after the determination of the desired processing path and before the junction point is reached — destroys or alters the data. For example.

(1) Set "No Operation" in variable connector 6

(2) If X = B, set "Jump to step 12 in 6"

(3) If X = C, set "Jump to step 19 in 6"

(4) If X = D, set "Jump to step 25 in 6"

(5) Set X = Y (intermediate processing)

(6) Variable connector.

(7) Processing if X ≠ B, X ≠ D.

(8) .

. .

. .

(12) Processing if X was equal to B.

(13) .

. .

. .

(19) Processing if X was equal to C.

(20) .

. .

. .

(25) Processing if X were equal to D.

The concept of variable connectors is a most important programming tool; however, it may be effected only in GAP by 1604-B instructions to perform the set instructions. This requires the usual (or perhaps unusual) synchronization of the 1604-B and the GAP.

7.7 QUERIES

7.7.1 Command List Form

The preceding paragraphs dealt with the required executive type and auxiliary functions needed to prepare a query into processable form and to obtain its required inputs. This form which is the command list is executed interpretively by the Run Routine. The queries given in Paragraph 3.5 are presented in a pre-processor command list form in Table 7-1. The data base files and directories required by each query for input are listed along with an indication of the output.

TABLE 7-1. COMMAND LIST REPRESENTATION

Question	Step	Opcode	Command List		Data Input	Output List
			L$_1$ (address of)	L$_2$ (address of)		
How many ships are within "r" miles of point "p"?	(1)	DIST	LAT/LONG of point "p"		Dynamic Ship File Directory	Registry/SSN DIST distance } for each ship in directory
	(2)	REL5	DIST	"r" in nautical mi.	STEP (1)	Registry/SSN
Are any U.S. submarines in area_________?	(1)	REL1	Ship Type	SUBMARINE CODE	Static Ship File	Registry/SSN
	(2)	DIST	LAT/LONG of area		Dynamic Ship File Directory	Registry/SSN DIST distance } for each ship in directory
	(3)	REL5	DIST	# mi. (defined as within an area)	STEP (2)	Registry/SSN
	(4)	AND	STEP (1)	STEP (3)	—	Registry/SSN
How soon can DD-789 reach Bermuda under normal SOA?	(1)	REL1	Registry/SSN of DD789		Dynamic Ship File Directory	Record of DD-789 logical name - lat/long.
	(2)	TIME	LAT/LONG at Bermuda		STEP (1)	TIME time for DD789
Where is Admiral_____ now?	(1)	EQU	Name — Admiral ____________		Personality File Directory	Registry/SSN
Are any ships scheduled to be in the projected vicinity of Typhoon Dottie the next few days?	(1)	DIST	LAT/LONG of Typhoon		Dynamic Ship File Directory	Registry/SSN DIST distance } for each ship in directory
	(2)	REL5	DIST	# mi. (defined as within vicinity)	STEP (1)	Registry/SSN

TABLE 7-1. COMMAND LIST REPRESENTATION (CONT.)

Question	Step	Opcode	Command List		Data Input	Output List
			L$_1$ (address of)	L$_2$ (address of)		
Nearest (in time ship/aircraft with aircraft aboard with doctor to point x, y?	(1)	REL1	SHIP TYPE	AIRCRAFT	Static Ship File	Registry/SSN
	(2)	REL1	Doctor on Board	YES	Oper. Char. File	Logical names
	(3)	EQU	STEP (2)	—	Oper. Char. File	Registry/SSN
	(4)	AND	STEP (1)	STEP (3)	—	Registry/SSN
	(5)	EQU	STEP (4)		Static Ship File Directory	All related logical names
	(6)	REL1	STEP (5)		Dynamic Ship File Directory	Directory records
	(7)	TIME	LAT/LONG of x, y		STEP (6)	Logical name — lat/long TIME time
	(8)	MIN	TIME		STEP (7)	logical name — lat/long
	(9)	EQU	STEP (8)		Dynamic Ship File Directory	Registry/SSN
What ships with long-range radar could be in area ________ by 2400 tomorrow?	(1)	REL1	Long range radar	YES	Oper. Char. File	Logical names
	(2)	EQU	STEP (1)	—	Oper. Char. Dir.	Registry/SSN
	(3)	EQU	STEP (2)	—	Static Ship File Directory	All related logical names
	(4)	REL1	STEP (3)	—	Dynamic Ship File Directory	Directory Records
	(5)	TIME	LAT/LONG of Area		STEP (4)	Logical name lat/long TIME time } for each record
	(6)	REL5	TIME	# hrs. to 2400 tomorrow	STEP (5)	Logical name — lat/long
	(7)	EQU	STEP (6)	—	Dynamic Ship file Directory	Registry/SSN
Ship sinking --- Which ships can be there first?	(1)	TIME	LAT/LONG of sinking ship		Dynamic Ship File Directory	Logical name — lat/long TIME time logical name— lat/long
	(2)	MIN	TIME		Step (1)	

TABLE 7-1. COMMAND LIST REPRESENTATION (CONT)

Question	Step	Opcode	Command List		Data Input	Output List
			L_1 (address of)	L_2 (address of)		
Nearest ocean-going tug to sinking ship.	(1)	REL1	Ship Type	TUG	Static Ship File	Register/SSN
	(2)	REL1	STEP (1)		Dynamic Ship File Directory	Directory Records
	(3)	DIST	LAT/LONG of sinking ship		STEP (2)	Logical name — lat/long DIST distance
	(4)	MIN	DIST		STEP (3)	Logical name — lat/long
	(5)	EQU	STEP (4)		Dynamic Ship File Directory	Registry/SSN

7.7.2 Timing Considerations

One initial objective of this study was to compute the time requirements for every hybrid configuration considered, and compare these times to those times (to be obtained from another source) for a similar problem effected on a system which did not include an associative memory. As the study progressed, it became apparent tnat the timing information from the outside source would not be available. It was then decided to encode certain sections of our solution to develop these times. The required usage of project time and manpower was at the expense of developing timing information for parts of our systems designed for the different configurations. In particular, explicit timing expressions for the translator, pre-processor, controller, and input routine do not exist. On the other hand, the algorithms given for these subsystems are of sufficient detail that implicit timing statements may be made. The timing for a particular query may be considered to be the time required to:

(1) Translate the query into Polish prefix form.

(2) Pre-process the query into command list form.

(3) Read in the required inputs.

(4) Perform the subroutines listed in the command list — the Run Routine.

(5) Output the results.

7.7.2.1 Subroutine Timing. The timing required for each subroutine given in Paragraph 7.4.1 for the Goodyear Associative Processor, the 1604-B, and the A3 associative processor is given in Table 7-2. The timing formulas for the Goodyear Associative Processor assume no input-output operations and use the average elapsed instruction times. The formulas for the 1604-B assume average 1604-B instruction times. The timing requirements given for the Goodyear Associative Processor are divided into two main classes: processing with no other I/O operations and processing with moderate (other) I/O operations. Additionally, the timing requirements for both the Goodyear and the A3 associative processors are given as the sum of the times required to load the associative memory and to do the processing. The Move instruction referenced with the A3 times refers to a multi-word 1604-B Move instruction which has been estimated to require 4.6 microseconds per word transferred as compared with 18.8 microseconds per word without this instruction. The Move instruction is one which can be imple-

TABLE 7-2. SBUROUTINE TIMING COMPARISON (Milliseconds)

SUBROUTINE	GAP						1604-B	A3				
	AM Load Time		Proc. Excl. AM Load Time		Total			AM Load Time		Proc. Excl. AM Load Time	Total	
	No I/O	Mod I/O	No I/O	Mod I/O	No I/O	Mod I/O		With Move	Without Move		With Move	Without Move
REL	7.3	21.0	2.0	4.0	9.3	25.0	5.9	4.7	19.3	2.0	6.7	21.3
AND	1.4	5.1	0.9	1.9	2.3	7.0	1.7	0.9	3.8	0.9	1.8	4.7
OR	1.4	5.1	0.9	1.9	2.3	7.0	1.9	0.9	3.8	0.9	1.8	4.7
MAX(MIN)	7.3	21.0	0.5	1.1	7.8	22.1	5.7	4.7	19.3	0.5	5.2	24.3
NGT(NLT)	7.3	21.0	0.5	1.1	7.8	22.1	8.1	4.7	19.3	0.5	5.2	24.3
EQU	7.3	21.0	3.6	7.4	10.9	28.4	7.1	4.7	19.3	3.6	8.3	22.9
TIME	7.3	21.0	12.0	14.0	19.3	35.0	15.0	4.7	19.3	12.0	16.7	31.3

mented in the 1604-B. From the timing presented in Table 7-2 it may be seen that it enhances the A3 memory.

7.7.2.2 Query Timing. The queries given in Paragraph 3.5 were converted to command list form as shown in Table 7-1. Then, the timing of each query was effected and is shown in Table 7-3. The step numbers in the table refer to the steps given in the command list (Table 7-1). To provide timing, it was necessary to assign values to the parameters found in the timing formulas. The following assumptions were therefore made.

(1) There are 32 records in a data file.

(2) A data record contains on the average 32 words.

(3) The average number of responders per search is four.

(4) There are 200 records in a file directory.

(5) A file directory record contains on the average five words.

7.7.3 The Non-Associative Memory System

Comparison of the instruction repertoire and timing between the two systems has highlighted several important observations. It is interesting to note that the execution time of the query subroutines is faster when they are performed in the 1604-B than in the hybrid system. This is because of load time. It takes 7.1 microseconds per word to load the AM from core, while a search performed in core takes on the average 3.6 microseconds per number of words searched. This also holds true in the A3 system where core-to-core transfer time consumes 18.8 microseconds per word without the Move instruction. Examination of the timing of the representative questions further substantiates this observation. In almost all steps, 1604-B execution time used less than the GAP and A3 time. It is interesting to note, however, that in step (3) of the query (in ιable 7-1) "Nearest (in time) ship/aircraft with aircraft aboard with doctor to point x, y?", the GAP time was faster. This was because it was not necessary to reload the AM.

It should be mentioned that the file structure and data ordering assumed in the non-associative memory system are that file structure and data ordering given for the hybrid configurations. If the former (non-AM) system were to be effected, the file structure and ordering would be changed to a structure and ordering that is more amenable to non-AM processing. Indeed, the AM structure and ordering are perhaps the "worst case"

TABLE 7-3. TIMING OF REPRESENTATIVE QUESTIONS* (Milliseconds)

QUESTION	STEP	GAP		1604-B	A3	
		No I/O	Mod I/O		With Move	Without Move
How many ships are within "r" miles of point "p"?	(1)	352.0	352.0	352.0	352.0	352.0
	(2)	9.3	25.0	5.9	6.7	21.3
	TOTAL	361.3	377.0	357.9	358.7	373.3
Are any U.S. submarines in area ___________?	(1)	9.3	25.0	5.0	6.7	21.3
	(2)	352.0	352.0	352.0	352.0	352.0
	(3)	9.3	25.0	5.9	6.7	352.0
	(4)	2.3	7.0	1.7	1.8	4.7
	TOTAL	372.9	409.0	365.5	367.2	399.3
How soon can DD-789 reach Bermuda under normal SOA?	(1)	9.3	25.0	5.9	6.7	21.3
	(2)	19.3	35.0	12.0	16.7	26.6
	TOTAL	28.6	60.0	17.9	23.4	47.9
Where is Admiral________ now?	(1)	10.9	28.4	7.1	8.3	22.9
	TOTAL	10.9	28.4	7.1	8.3	22.9
Are any ships scheduled to be in the projected vicinity of Typhoon Dottie the next few days?	(1)	352.0	352.0	352.0	352.0	352.0
	(2)	9.3	25.0	5.9	6.7	21.3
	TOTAL	361.3	377.0	357.9	358.7	373.3
Aircraft or ship, with doctor aboard, nearest in time to point x, y?	(1)	9.3	25.0	5.9	6.7	21.3
	(2)	9.3	25.0	5.9	6.7	21.3
	(3)	3.6	7.4	7.1	3.6	3.6
	(4)	2.3	7.0	1.7	1.8	4.7
	(5)	10.9	28.4	7.1	8.3	22.9
	(6)	9.3	25.0	5.9	6.7	21.3
	(7)	55.3	62.7	51.0	52.7	67.3
	(8)	7.8	22.1	5.7	5.2	24.3
	(9)	10.9	28.4	7.1	8.3	22.9
	TOTAL	118.7	231.0	97.4	100.0	209.6
What ships with long-range radar could be in area _________ by 2400 tomorrow?	(1)	9.3	25.0	5.9	6.7	21.3
	(2)	10.9	28.4	7.1	8.3	22.9
	(3)	10.9	28.4	7.1	8.3	22.0
	(4)	9.3	25.0	5.9	6.7	21.3
	(5)	55.3	62.7	51.0	52.7	67.3
	(6)	9.3	25.0	5.9	6.7	21.3
	(7)	10.9	28.4	7.1	8.3	22.0
	TOTAL	115.9	222.9	90.0	97.7	199.9

* Note: Timing does not include disc unload time. See Paragraph 7.6.2 for list of assumptions.

structure and ordering for non-AM processing. In other words, the time requirements given for non-AM processing may be improved. Thus, 1604-B executive times may be decreased further. It is then apparent that the non-AM system is a better system if time requirement is a prime measure.

7.7.4 Frequency and Instruction Count

Tables 7-4 and 7-5, comparing frequency and count of instructions,are somewhat indicative of the relative programming ease* found in the associative memory compared with the non-associative memory. From this table (Table 7-4 for the 1604-B and Table 7-5 for the Goodyear Associative Processor), it may be seen that it takes on the average twice as many 1604-B instructions to perform a subroutine than the number of instructions required in a hybrid system. This reduction in the number of instructions is chiefly attributed to the powerful search instructions of the associative memory. On the other hand, the structure and ordering of the data, which is biased to associative memory processing, increase the 1604-B instruction count over what it would be if this structure and ordering were oriented to non-AM processing

7.7.5 Timing of the Input Routine

The timing requirements given in the preceding paragraphs do not include the times required to input and output the results. The design of the query system was motivated by the desire to remove much of the bias imposed by the disc system by minimizing disc access times. This was accomplished by noting that the desired disc addresses of input blocks were known, a priori, and could be ordered prior to accessing such blocks. This fact is true in any system, hybrid or not. Indeed, the input times for both systems would be identical if the same data base organization and the same disc system were used. This follows because the physical organization of a data base is more dependent upon the disc file than upon the processors (The ordering of data within accessed blocks is more dependent upon processors than upon the disc file).

The data access time for the CDC 818 is equal to the sum of the positioning, confirmation, and latency times (Paragraph 7.6.1). The maximum is 354 milliseconds (nominal, depending upon confirmation time); the average is 225 milliseconds. After accessing the desired block, data transfer takes place. This transfer may be given on a word rate

*See Paragraphs 7.3.5 and 7.6.3.1.

TABLE 7-4. 1604-B FREQUENCY AND NUMBER OF INSTRUCTIONS PER SUBROUTINE

Instruction	SUBROUTINE NAME						
	REL	AND	OR	MAX	NGT	EQU	TOTAL
Data Transmission							
LDA	6	1	2	3	10	2	24
STA	2	3	3	1	3	2	14
Shifting							
ARS	-	2	2	-	-	-	4
ALS	1	1	1	-	1	1	5
LRS	1	-	-	-	-	-	1
Address Modification							
SAU	5	2	2	8	9	3	29
IJP	1	-	-	-	-	-	1
ISK	1	1	1	1	3	1	8
SIU	-	-	1	-	-	-	1
Arithmetic							
SUB	1	1	1	1	2	1	7
No Address							
ENQ	3	3	2	2	3	3	16
ENA	5	3	3	6	9	6	32
INA	2	1	1	2	3	1	10
ENI	5	4	3	3	5	3	23
INI	3	1	2	2	5	1	14
Jumps							
AJP	1	1	1	1	1	-	5
SLJ	5	2	4	3	11	4	29
QJP	2	-	-	-	-	-	2
Logical							
LDL	2	2	2	2	2	-	10
ADL	1	1	1	-	1	1	5
Storage Search							
EQS	2	1	1	1	4	1	10
THS	1	-	-	-	2	-	3
MEQ	-	-	-	-	-	2	2
Replace							
RAO	1	-	-	-	-	2	3
TOTAL	51	30	33	36	74	34	

TABLE 7-5. GAP FREQUENCY AND NUMBER OF INSTRUCTIONS PER SUBROUTINE

Instruction	SUBROUTINE NAME						
	REL	AND	OR	MAX	NGT	EQU	TOTAL
External Functions							
RESUME	1	1	1	1	1	1	6
CLEAR	1	1	1	1	1	1	6
Load							
LDR	2	2	2	4	5	2	17
I/O							
RBL	1	1	1	3	3	1	10
RCR	-	-	-	2	2	-	4
RDA	1	1	1	3	3	1	10
Erase							
EMY	1	1	1	3	3	1	10
Index							
ICI	1	2	2	1	1	1	8
LDI	-	-	1	-	-	-	1
SIX	-	1	1	1	1	-	4
Control							
JUC	-	-	-	1	1	-	2
JNR	2	1	1	1	1	-	6
JIH	1	1	1	2	2	1	8
Search							
EMC	2	1	1	1	1	1	7
MMC	1	-	-	-	-	-	1
LTC	1	-	-	-	-	-	1
GEC	1	-	-	-	-	-	1
LEC	1	-	-	-	-	-	1
GTC	1	-	-	-	-	-	1
MAX	-	-	-	1	-	-	1
MIN	-	-	-	1	-	-	1
NLC	-	-	-	-	1	-	1
NHC	-	-	-	-	1	-	1
1604B							
LDA	2	1	1	1	1	1	7
LDQ	1	-	-	-	-	-	1
MEQ	1	-	-	-	-	-	1
ENA	1	-	-	-	-	-	1
AJP	1	1	1	-	-	-	3
THS	1	-	-	1	1	1	4
STA	1	-	-	1	1	1	4
SUB	-	1	1	-	-	-	2
TOTAL	20	16	17	28	28	13	

as follows: 112 microseconds on the inner zone and 68 microseconds on the outer that is, an average of 90 microseconds.

The ordering given by the Controller reduces the given access times. Su reduction depends upon the classes of data retrieved; that is, how many blocks are referenced from the same position of the same disc. From Paragraph 7.6.1.1, ite (2), where the data allocation is given, it is seen that data from the same class is stored the same disc. From this fact, assuming (as in Paragraph 7.7.2.2) 200 directory r ords of five words each for every variable, and assuming there are about 15 blocks per p ion that must be retrieved for each variable, the access time requirements (Paragraph 1) are greater than 1122 milliseconds. The data transfer times and access times then about 1.2 seconds/variable on the average. Thus, the query "Nearest (in time) ship aircraft with aircraft aboard with doctor to point x, y." would have an additional 7.2 onds added. This is 35 times greater than the processing time required. Thus, even with sophistication given by the Controller and the Input Routine to minimize these times, system is still input bound for every query.

In this respect, then, a natural choice is to consider a different disc syste for example, the IBM 1302. Pertinent characteristics regarding input rates follow.

7.7.6. IBM 1302 Disc Unit

The 1302 has two magnetic disc modules on a common vertical shaft with c cities of 117,000,000 six-bit, or 70,660,000 eight-bit characters per module. Each moc consists of 25 metal discs coated on both sides with magnetic oxide. The discs are spaced to provide access for the read-write head access mechanisms. A small magn slug, mounted on the periphery of the format disc, is sensed by magnetic transducers provide index signals for timing reference.

Each module is addressed by two access mechanisms, each having 24 arms which move radially in the spaces provided between the discs. Two read-write heads mounted on each arm. One of the heads services the bottom surface of the disc above the arm while the other head services the top surface of the disc below the arm. In th way, it is possible to read or write on either side of a disc.

An access mechanism, driven by a hydraulic system, is capable of assuming 250 distinct radial positions. For each module, one access mechanism services the outer 250 positions; the other access mechanism services the inner 250 positions. For each of the two access mechanisms any one of the 250 distinct radial positions assumed by a head defines a data track. Thus, on one surface of a disc there are 500 tracks. The 50 disc surfaces in a module are used as follows: 40 for data storage; one for clock tracks; one for format tracks; six alternate data surfaces; two (the top and bottom external surfaces) are not used. At each of the 500 access positions then, there are 40 data tracks available by merely switching heads. These 40 data tracks, aligned vertically, define a cylinder. Each 1302 module contains 20,000 data tracks (40 tracks x 500 cylinders), 10,000 serviced by one access mechanism and 10,000 serviced by the other access mechanism. Neither access mechanism may service tracks of the other access mechanism.

The discs rotate at a rate of 1,790 revolutions per minute or 34 milliseconds per revolution. This yields an average latency time of about 17 milliseconds. The time required to change access mechanism position ranges from 50 milliseconds for small changes to 180 milliseconds for large changes. Therefore, maximum access time is 214 milliseconds (180 + 34). The four access mechanisms in the 1302 are positioned independently but use a common hydraulic system.

Internal timing is provided to yield transfer rates of approximately 234,000 six-bit characters per second or 181,320 eight-bit characters per second. Assembly of eight 6-bit characters per 48-bit 1604 word yields a transfer rate of 29,200 words per second. Space allotted for addresses and gaps on the data tracks reduces the effective transfer rates. A gap is the space provided to separate two data areas. An entire cylinder of data (234,000 6-bit characters) can be transferred in approximately 1.33 seconds.

Two internal control panels are provided in the logic section of the 1302. One contains a group of switches and indicators for the power on-off sequences. The other provides controls and indicators for the off-line maintenance. Both of these panels will normally be used only for maintenance. A format read-write key-lock switch for each module is located beside the disc array. It is accessible only with the 1302 cabinet door open.

Thus, the IBM 1302 compared with the 818 requires from 67 to 214 milliseconds for each access, and with one access can transfer data at a word rate of 34 microseconds for a total of 234, 000 6-bit characters. The average access time and the data transfer rates are less than 30 percent of the 818 requirements. Additionally, more than seven times the volume of data may be retrieved with one access. The overall effect is to reduce the time requirements for data input on approximately the same magnitude as the processing requirements (recall that data must be transferred to the associative memory before it can be processed; that is, the disc cannot be read directly into the associative memory except, of course, the A3 memory).

Perhaps the biggest impact of the 1302 on a problem of the type would arise from use of the format track. This is explained as follows:

> Before the disc module can be used for reading or writing, a format track must be written for each cylinder of the module. The format track permits the programmer to designate, within certain limits, how the storage space of the data tracks of a cylinder is to be allocated, identified, and used. Once established, the format track provides a fixed format and control for the subsequent reading or writing of data for that cylinder. All data tracks within a cylinder must have identical formats. The format for a cylinder may be altered at any time by re-writing the format track.
>
> Data used to write a format track must first be organized in memory as a data record. An appropriate instruction is then issued to the 7631 and the data is transferred to the addressed format track. Once written, the format track remains unchanged until re-written. The format track is used in conjunction with variable length records on each data. Consider the impact on maintenance, where record sizes were adjusted to a fixed number of words.

7.8 OUTPUT FUNCTIONS

After all the elements in the command list have been processed, the Run Routine transfers to an output subroutine which performs one or more (the last element in the command list contains this information) of the following output operations:

(1) "Yes and No" depending on whether or not one or more items satisfy the condition (i. e., respond).

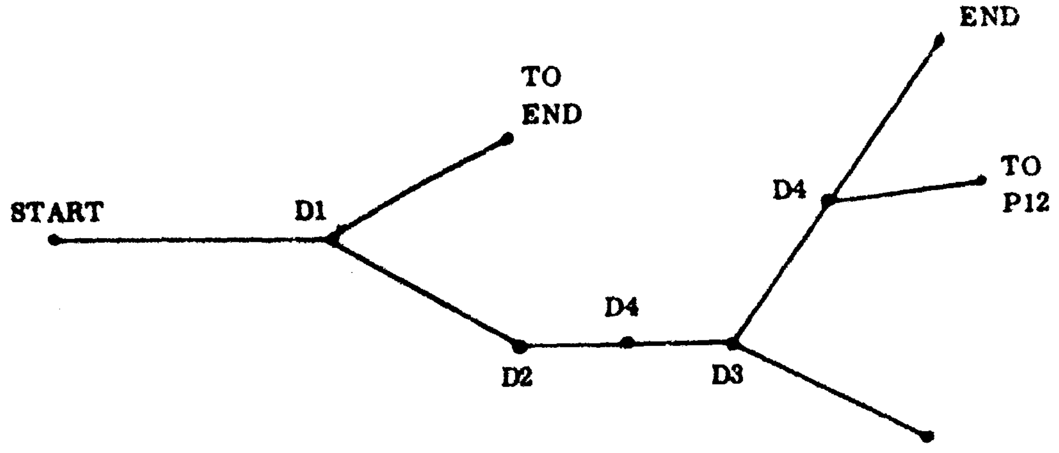
(2) "Count" the number of items which respond (number of responders).

(3) "List" the logical names of the responders or the record content of the responders.

Each of these output operations may also be considered as an element of the command list. In every associative memory considered in this study, the first two functions (which must be representable by a subroutine) may be performed by the instructions "Jump on No Responders" and "Count Responders," respectively. The last function may be performed by executing the first two (to obtain N for the output function) then transferring control to a 1604-B output routine.

7.9 TAG MEMORY

Most of the processing required to translate and pre-process a query in Polish prefix form into a canonical command list form is data dependent processing. In addition, the processing required to control (Controller Routine) the Run Routine and the Input Routine contain a significant amount of data dependent processing. Data dependent processing may be explained by considering decision points D_i $(1 \leq i \leq K)$ in a program. For each decision point, let there exist more than one path P_{ij} $(1 \leq j \leq L)$. The particular path to be taken may be decided at the processing time the decision point is reached. This implies that the origin of the path P_{ij} are coincidental with the decision point D_i. Additionally, the beginning junction of the paths P_{ij} may be "set" by D_i at some processing time which precedes the time of following one of the paths. In this case, the junction point is known as a "variable connector" (Paragraph 7.6.3.1). Each path P_{ij} may be defined to connect two decision points D_i, or one decision point and one variable connector, or two variable connectors. For example:



In this example, the decision points D_1, D_2, and D_3 determine one of several (in this case) processing paths to be followed. Notice that the decision point D_4 which occurs in Path P_{23} (between D_2 and D_3) "sets" the path at the point D_4. This is an example of a variable connector.

If a processing path from start to end (P_{s1}, P_{12}, P_{23}, P_{34}, P_{4E}) performs decision points in a serial fashion, the conditionals or decision points are said to be serial. Then, nested conditionals occur when decision points are repeated; for example, the processing which results from repeatedly following the path P_{41}.

It was pointed out in Paragraphs 7.3.5, 7.6.3.1, for example, that nested conditionals and variable connectors were difficult to handle in the Goodyear Associative Processor. In this respect, the main advantage of associative memories is the determination of the status of a large amount of data with one instruction in the "same" time usually required to determine the status of one element of the data in a conventional memory. The results of the Search instruction in the associative memory are stored in a "response store." However, in the Goodyear associative memory, there exists little direct utilization of this response store.

A need occurred frequently to retain a response store for future processing. Since no provision existed directly to do this, the first approach (Paragraph 7.6.3) is to store the data in one half of the associative memory and use these instructions to save a response store (at the expense of 1,024 storage words);

MMC 1,0,0,0	RESET D and RS
MMC 2,0,0,0	RESET E " "
EMC 1,0,0,0	SET D " "
EMC 2,0,0,0	SET E " "
EMC 1,0,1,0	Copy RS into D
EMC 2,0,1,0	" " " E
EMC 1,0,1,1	AND " " D
EMC 2,0,1,1	" " " E
EMC 1,0,1,2	OR " " D
EMC 2,0,1,2	" " " E
MAX 1,0,0,1	Copy D " RS
MAX 2,0,0,1	" E " "

The disadvantage here is that only past and one current response store may be utilized and that one half of the memory is "lost." To remedy this situation, as suggested in Paragraph 7.6.3, let there exist an association between data words in the lower half of the memory and tag words in the upper half. This association depends upon the structure of the data in the lower half. One immediate (and constant) association is given in that the addresses are both modulo 1024. If each data element is a member of the same set, for example, logical names, then this address association may be sufficient. But suppose that the data consists of several different sub-elements all related to one element and that there are many such elements. For example, suppose the elements designate variables and the sub-elements are such things as the 1604-B storage location of the variable, the norm of the variable, etc., as in the command list, then implicit address association previously noted is insufficient. The desired association is accomplished as follows:

Let there be M classes of elements E_m $(1 \leq m \leq M)$ such that class E_m is a sub-element of class E_m. Let there exist N_m sub-elements in the m^{th} class. Then the elements are more correctly labelled as E_{mn}; $(1 \leq n \leq N_m$; $m = 1, 2, \ldots, M)$. Assume that these data elements are stored in the lower half of the associative memory. For each such data word stored at address A_L in the lower memory, there exists a tag word in the upper memory at address $A_u = (A_L + 1023)$. This addressing is implicit and obtainable by use of P element in the associative memory instruction. Divide each tag word in the upper half of the memory into two parts, A and T, of 10 and 38 bits, respectively.

The element in A_L is some E_{mn}. In the A portion of $A_u = (A_L + 1023)$ place the lower memory address of the element E_{ml}. In the A portion of the tag word corresponding to E_{ml}, place the lower memory address of the element $E_{m+1,1}$. Thus, the address tags of the elements E_{mn} $(2 \leq n \leq N_m)$ and $E_{m-1,1}$ are the lower addresses of element E_{ml}.

The element T of each tag word is used to reflect the status of the element associated with it, not its leading class number. The programmer must decide a priori what bit or bits are allocated to each status and what the setting of the bits means. Suppose, for example, that the first bit of T indicated the condition of implicit or explicit variables. The data in the lower memory is tested for this condition. The D register is then transferred to the E register and a full word of ones (111 . . . 111) is written into responders through a mask with one and only one bit in that position corresponding to the first bit of T.

It is therefore feasible using this Tag memory to save at least 38 previous conditional test results (response registers) by appropriate manipulation of the mask register.

Since addresses of responders in the lower memory are stored in the upper memory,it is possible to query the range of addresses for responses as well. This device eliminates the need, or may be used to obviate the need,to shift buffers in processing fixed field data.

This concept of Tag memory (which is very similar to the A2 associative memory) removes the inability of the programmer to store, retrieve, manipulate, combine, etc. previous response store buffers. This is felt to be a most important concept in the use of the Goodyear Associative Processor.

SECTION VIII. COMPARISON OF THE A2 AND GAP ASSOCIATIVE SYSTEMS

8.1 INTRODUCTION

This section describes how GAP and A2 functions compare to one another. This description is presented on a functional level; the comparison analysis is accomplished using GAP instructions.

The motivation behind the comparison of one memory system to another can best be understood by noting that A2 is approximately 40 percent less costly than GAP, and that in the given hybrid configuration, the timing of the sea surveillance system is more nearly a function of disc processing time than of associative memory processing time. In light of these two considerations, it is natural that the conjecture would arise that an economical memory, on the order of A2 but lacking in some of the more sophisticated features of GAP, might well prove more than competitive. This conjecture proves to be basically true, provided that considerable attention is given to all the qualifications that must be made in conjunction with it.

As noted in Sections V and VII, the use of a Tag memory and the use of a memory which permits ready use of response vectors by the programmer is preferable for many programming tasks. A2 has ample response vector and tag bit capability. The method of testing the stated conjecture, i. e., comparing one memory to another in terms of GAP instructions, tends to almost completely obscure this important aspect of the A2 memory. For this reason, the method of comparison biases the results in favor of GAP. In light of the time available for the study, the method was, nevertheless, preferable because full utilization of the tag bit and response vector capability of the A2 memory would have compelled a redesign of many system software components.

It must also be noted that the comparison of GAP to A2 cannot be completed in a full sense because the memories are organized differently, they communicate with the central processor differently, and they reflect different ideas about the use of associative processors. These differences, detailed in Section II, compel us to employ the concept of comparing the functions of the different instructions applicable to the memories.

The comparison is presented in two levels: general and detail. Loading and unloading are treated in a general level of comparison because they are limited by their respective memory configurations. Detailed comparisons are given for searches; P, T, V, and Z tags; and control operations.

The method of comparing some instructions is not necessarily optimized. Efforts to obtain efficient code were limited to those instructions of the GAP that were known to be used quite often. In the area of programming, it may be quite possible for the comparison to be favorable to the A2.

It is important to note that the method of comparison selected for this section ignores certain features of GAP that might be of considerable value in dealing with some problem other than the sea surveillance problem. For example, the ability of GAP to work in parallel with the 1604-B is ignored. This feature was found to be of little value in the problem at hand, although it is possible that other problems might find it important.

8.2 REDUCTION OF GAP TO A2

8.2.1 Communication of AM Conditions

GAP provides for external functions to sense conditions prevailing in the associative processor. A2 provides instructions to perform the same tasks in the same manner. The following table therefore provides proof of comparison:

GAP Function	A2 Instruction
(1) Sense-Activity	TRA (Sense alive)
(2) Sense-Parity	TRE (Sense error)
(3) Sense-Overflow	TRO (Sense overflow)
(4) Clear	CAM (Clear AM)

The fact that this parallelism exists simplifies the comparison but ignores the fact that, for two important conditions, overflow and error, the "sense" option is far less preferable to the A2 programmer than other options of the A2. In actual programming, the A2 programmer is more likely to use the interrupt capability of the machine than the sense instructions alone. Thus, for parity conditions, the programmer is likely to have used the "IER" instruction to tell the AM to interrupt on an error. In this event, he need not code regular sensing sequences, or entries to them, after performing AM operations. Instead, he could rely on the interrupt to transfer him to his abnormal condition routine when and only when the error occurs. Similarly, he would use the "IOV" instruction to generate an interrupt whenever an overflow condition arose.

Since overflow and error are the most important AM conditions for the working programmer, the interrupt capability of the A2 will make the task of coding simpler. It should also be noted that the design of an AM executive is an easier task for A2 than GAP. A2 is therefore more convenient to use for overflow and error.

With respect to the timing of operations to sense conditions, A2 and GAP should take nearly the same time, since both are essentially 1604 external functions. With respect to locating precisely what causes an error or overflow condition,the time required in A2 should be significantly less than in GAP since the interrupt will signal when the error occurs, while in GAP the programmer is likely to be testing (sensing) conditions only after an entire AM coding sequence has been performed. No specific measure of this time is possible, of course, without actually coding each specific case and developing formulas for each.

8.2.2 Force and Resume

The two 1604 external functions which either force or resume AM processing for GAP do not exist for A2. For this reason it is necessary to discuss the functions performed by the force and resume instructions in the problem at hand. The following discussion ignores the parallel processing capability of GAP.

8.2.2.1 Force. The external function to force the AM normally appears in a sequence of 1604-B coding designed to cause the AM to begin the execution of GAP coding. The "force" causes the AM to fetch an instruction from location 77777_8, which will normally contain a "JUC," or "jump unconditional" command. This command will transfer control to a routine stored in the 1604-B for execution by GAP. The only other use contemplated for the force in the problem at hand (sea surveillance) is to cause the AM to stop. In this event, location 77777_8 contains a "HLT" or "halt all AM operations." Each of these cases is treated separately.

When the force is used to cause transfer of control, it appears embedded in 1604 coding, and is probably preceded by at least one sense instruction. Location 77777_8 contains a "JUC" to transfer control to the AM coding to be executed. The following table therefore shows the sequence and timing involved:

Instruction	Timing (μ sec)
SEN (to prepare for EXT)	7.0 (avg)
EXT (to force the AM)	7.0 (avg)
JUC (GAP coding to transfer control)	20.5
Total	34.5

In the A2, this sequence can be replaced by at most,one instruction, and usually would require no instruction at all.

In the event that the programmer desired simply to perform steps in the A2 associative memory, no sequence is required other than the coding sequence to be performed. In the event that the programmer desired to perform a subroutine stored elsewhere in the 1604 memory, either a return jump or a selective jump would suffice, Thus, while either the force or resume will be required in the GAP system whenever AM processing is desired, it is always possible to enter processing directly using A2. This is true because all instructions in the A2 are either EXT instruction codes or bit codes transferred to the I register of A2 by EXT instructions. (Loading or unloading data can also include INT or OUT instructions.)

In the second usage of the force, in which the force causes a halt, there is nothing analogous in A2. The value of a halt in GAP has proved to be the value of stopping the AM processing with the contents of registers and the instruction counter intact. This will always be the case with A2. The halt also causes the "AM active" line to be dropped. Since A2 functions as a peripheral device, the A2 is inactive at the completion of every operation.

8.2.2.2 Resume. Like the force, resume is used to initiate processing by GAP. Resume differs from a force in that the present value of the instruction counter is taken as the place to begin AM processing, rather than location 77777_8. A resume will probably always be preceded by a sense instruction, or by several sense instructions. In anv actual task in the problem under discussion, however, it is always possible to replace a resume with either a selective jump or a return jump. In most cases, no instruction will be required. Once again, the A2, because it is essentially a unit on a transfer channel, does not require the special start-up instructions of the GAP. Of course, it lacks the parallel processing capability as a result..

8.2.2.3 Conclusions on Force and Resume. Because of the way in which A2 works, neither the force nor the resume are required. Consequently, in A2, the time required for the force and resume operations is saved. Because a force or resume sequence is required to initiate every entry into associative processing with GAP, this difference is a significant one. Where n is the number of times that AM coding must be entered from 1604-B only processing,and where E is the average time required to execute an external function under the input-output load of the problem at hand, n(E-7.2) microseconds is required for either force or resume operations in the cases most favorable to GAP.* A more likely condition for the problem at hand is n(2E), since no provision would be required for transfer of control using A2, and, as designed, it is very likely that at least one sense instruction would, of necessity, be required prior to each force or resume in a GAP software system.

8.2.3 P, T, V, and Z Tags.

The P, T, V, and Z tags are control tags that exist in GAP instructions. There is a rough correspondence between these tags and some of the fields of A2 instructions, although in fine points there are significant differences that make comparison difficult. Each is treated separately in subsequent paragraphs.

8.2.3.1 The P Tag. The P tag corresponds to the UL field of A2 instructions whenever it is used to specify that an operation is to be performed in one half of the memory only and whenever no logical operations with buffers are involved. There are, therefore, cases in which the UL fields will suffice for representing the data normally carried by the P tag. These cases, however, will prove to be relatively few in the sea surveillance system. Further, there is no simple mechanical procedure for the specification of the cases for which UL is a proper substitu for P. This situation arises because of the availability of the D and E buffers in GAP. Because the D and E buffers exist in GAP, GAP programmers can be carrying on one kind of processing in one half of the memory and another in the other, storing at least one vector in the buffers without confusion. In A2, if such flexibility is to be provided,1604-B memory must be used as temporary storage analogous to the D and E buffers.

In subsequent discussions, therefore, the UL field of the A2 instructions will not be used as a substitute for instruction data carried by the P tag. This decision causes some bias in favor of GAP, since there obviously are cases where the UL field would suffice. Nonetheless, the bias is minor, owing to the difficulty of identifying general rules.

* "E" has a value somewhere between 6 and 8 microseconds, tending toward the upper limit.

8.2.3.2 The T Tag. The A2 memory has the provision to shift the response store in both directions, and to shift this store a variable number of places. With the convention that buffers will be represented as blocks of 1604-B storage, and with the obvious need of maintaining these blocks of data, the use of the SHIFT field of A2 instructions, with SHIFT having the value of 1 whenever the T field requires a shift, will provide the needed simulation of GAP. It should be noted, however, that the BA field must always have the value B when a shift is required.

Simulation of the T field in A2 emphasizes the relative superiority of A2 with respect to shifting, since neither the BA field nor the SHIFT field can be used to fullest advantage in such a simulation. Because of the ease with which the T tag is simulated, no further discussion of it is required.

8.2.3.3 The V Tag. Full simulation of the V tag feature of GAP is simple enough once the convention to store vectors in the 1604 memory, as if they were buffers, is employed. There are two options available to the programmer to simulate the effect of the V tag: the tag bit option and the RS option. In the tag bit option, a particular tag bit position of the A2 is selected to represent the prior contents of S. Operations are then selected to use the tag bit in such a way that the final value of S will correspond to the final value of RS in GAP. This method is highly efficient for some operations but is hard to generalize, since the setting and use of the tag bit depend largely on the operation to be performed. In this comparison the RS option is employed.

In the RS option, a block of 32 words of 1604 memory is set aside to represent RS in GAP. Call this area RS1. The results of RS1 are "ORed" with the value of S whenever the GAP routine would use the V bit to inhibit the initial manipulation of response store. Only after RS1 and S are "ORed" are the buffer areas in the 1604 operat upon, depending on Z.

The V tag is clearly one more feature of added flexibility for GAP, as compar with A2. It is necessary, in this comparison, to set aside another core storage area in order to simulate this tag, in addition to the need to program the operations in question. It is interesting to note, however, that the V tag was little used in programming the sea surveillance system. Use of this tag in simulating certain tag memory operations in GAP has, however, been noted in Section VII.

8.2.3.4 The Z Tag. The Z tag controls logical operations to be performed between RS and the D and E buffers. The choices available to the programmer through using Z and the A2 instructions that correspond are:

Z Tag	A2 Instruction
Replace buffer with RS	RSR, reading S into 1604
"And" S and the buffer	Execute logic with computer
"Or" S and the buffer	" " " "
Leave results unaffected	No coding required.

It should be noted that this table will require the performance of one A2 instruction for three of the four options, and that 158 microseconds are required for each such instruction. In GAP these 158 microseconds are not required. In the problem studied, logic was frequently required for which the Z tag sufficed. In almost all cases, the use of the Z tag would require 158 microseconds per half memory in simulation using A2. For each such instruction 316 microseconds are even more likely, since the full memory option is most frequently specified. It can thus be seen that having the D and E buffers is advantageous for some standard operations.

8.2.3.5 The Role of SLS and Some Conclusions on Tag Fields. The SLS is an A2 instruction which, when preceding a search instruction, causes many of the tag simulation features of A2 to be performed automatically. When SLS is used in simulating GAP with the A2 memory, it becomes possible to simulate the effect of the T, V, and Z tags quite directly. The logic OP field of the SLS, for example, can be used to select in F_i operation which will cause "ANDing," "ORing," or direct transfer with the block storage being used to simulate the D or E buffer. Shifting can be specified at the same time, and the time required to execute the specified search will be at most on the order of 160 microseconds per half memory. For this reason, in searches, the full machinery of the previous discussion of the tag fields need not be employed. Simulation of the P tag continues to be difficult. In order to provide a convenient procedure for simulating the P tag, it is necessary to say that a sequence must, at the very least, be repeated twice when P requires that both halves of memory are acted upon.

Because of the simulation mode required for the tag fields of the GAP instructions, an increase should be expected in A2 processing over GAP processing in time over the range 158 to 320 microseconds for each operation performed using the tag fields. Since, in the sea surveillance problem, both halves of the memory have been used, the time differential is most likely to be 320 microseconds. Almost always, as will become clear later, this is the major time differential between GAP and A2 — considering software of the kind under examination in this study.

The P, T, V, and Z tags, and their usefulness, are one of the strongest recommendations for the GAP system of RS, D,and E, and the ability to perform logic on them.

Except when required for other reasons, the P, T, V, and Z tags will not be included in the following paragraphs discussing the comparison process, since the preceding paragraphs should provide an adequate guide for the programmer required to initiate GAP in an A2.

8.2.4 Index Operations and Control

GAP, because it can perform its operations in parallel with 1604-B operations, utilizes its own control, indexing, and indirect addressing. In using A2, which cannot operate in parallel in the same sense, the need for instructions to handle special AM indirect addressing, special AM indexing, and independent AM control does not exist. Consequently,existing 1604-B indexing, indirect addressing, and control features can be utilized by the programmer.

There are minor differences in the actual results of performing instructions whose usage is exactly parallel in GAP and the A2. For example, SIL of the 1604-B repertoire and SIX of the GAP instruction repertoire perform essentially the same task, except that at the completion of SIX, all parts of the affected memory location in higher order positions of the word are zero, while the SIL instruction portion of the word is not affected. Such minor differences can, by and large, be ignored, and it can be stated that GAP and certain 1604-B instructions and instruction sequences are functionally the same.

Instructions that are functionally the same, together with the corresponding average elapsed time for GAP and the execution times for the 1604,are:

GAP (μsec.)	1604 (μsec)
ICI 10.1	INI 3.0
SIX 14.2	SIL 7.2

LDI	14.2	LIL	7.2
DEI	10.1	INI*	3.0
JUC	7.1	SLJ	7.2
JIH	10.1	ISK	5.6
SPJ	14.2	SLJ	7.2
NOP	7.1	NO	8.0
GKT	7.1	None needed	

As can be seen, these operations can be performed more efficiently in the A2 1604-B than in GAP. Two GAP instructions require instruction sequences. These are JIL and JNR. For JNR, the following sequence is required:

Code	Comments	Time (μsec)
RCS	Read count of S	50 (average)
INT	Put count in 1604 memory	8
LDA	Load A	7.2
AJP	Jump if A = 0	7.2
		72.4 (average)

JNR requires approximately 7.1 microseconds for this operation, a ratio of 10 to 1 in favor of GAP. For JIL, the following steps might be used:

Code	Comments	Time (μsec)
SIL	Store index in memory	7.2
LDA	Load A	7.2
SUB	Compare with g, and if g > h,	7.2
AJP	Continue, otherwise go to h	7.2
		28.8

This compares with the 10.1 microseconds required for GAP. Hence, the GAP has almost a 3 to 1 superiority in this operation.

It can be concluded that (except for NOP, JIL, and JNR) control and index operations are simulated in the 1604-B/A2 configuration faster than in GAP, and that in this area the use of the peripheral device design of A2 appears sound.

8.2.5 Search Instructions

Three A2 search instructions appear to correspond exactly with GAP instructions: SOE, SOG, and SGE. This appearance is not completely correct. Handling of the GAP

* For DEI, use INI and a negative value; and for some DEI....JIL, use IJP.

tag fields (P, T, V, and Z) can require an additional sequence of operations requiring approximately 160 microseconds per half memory for simple cases, and 310 microseconds for more complex ones. In a completely general simulation of GAP by A2, these tag fields must be handled by routines which result in the following comparison:

GAP Operation	GAP TIME (μsec)	Basic A2 Operation	Estimated Maximum A2 Actual Operation Time (μsec)
EMC	32.2	SOE	368
GTC	32.2	SOG	368
GEC	32.2	SGE	368

A similar direct comparison is available for other GAP operations, noting that in A2, one can use the TF field to specify interest in those things which do not respond to a search. Hence, the following

GAP Operation	GAP Time (μsec)	Basic A2 Operation	Estimated Maximum A2 Actual Time (μsec)
MMC	32.2	SOE, f*	368
LTC	32.2	SGE, f*	368
LEC	32.2	SOG, f*	368

The instructions directly representable as in these two tables are the bulk of the GAP search instructions used in the sea surveillance problem. The times given above for A2 are estimated maximum times per 1 024 words searched. The bulk of the time included in this estimate is the time required to simulate the P, T, V, and Z tags.

Of the estimated maximum time for the searches in A2, 360 microseconds can be directly attributed to the need to simulate the effect of the P, T, V, and Z tags of GAP. In certain cases, it is possible to simulate the effect of these tags in a total maximum A2 time of only 8 microseconds, while quite frequently 166 microseconds are required. GAP requires a uniform average of 32.2 microseconds per search. The three A2 timing figures, 8, 166, and 360, represent three basic cases, of which only the 160 and 360 micro second cases are relevant to the sea surveillance problem. It can therefore be said that, for the GAP searches in question, GAP outperforms A2 in an order of magnitude of between 5 and 10 to 1 on pure search operations. It should

* The lower case "f" indicates that response store in GAP is represented as non-response in A2.

be noted, however, that nothing can be inferred from this concerning the relative efficiency of the memories to perform an actual processing operation involving a search. In actual coding, macros designed to locate and process data meeting a given search criterion utilize far more time performing other AM operations than they use for searching. In these other operations, A2 has a distinct advantage.

The more complex searches of GAP include CPX and related instructions, and the following:

GAP Instruction	GAP Time (μsec)	Estimated Maximum A2 Time (μsec)
MIN	20.3	528
MAX	20.3	528
BLC	43.1	400
NLC	39.5	400
NHC	39.5	400

The comparison of the instructions to A2 can be left to the reader.

The CPX, and its related instructions, SCF and END, cannot be timed except in terms of the actual operations that make up any given case. As is understood, CPX is simply a means to perform up to eight GAP searches within words using a different way to load the comparand, mask, and instruction registers of GAP. With this understanding, CPX can be simulated by A2 instructions as discussed above.

8.2.6 LDR and I/O

There is no need for LDR in an A2 system. LDR loads the values S, N, and R into the appropriate registers of GAP. In A2, shifting is done by the 1604-B and the number of INT or OUT instructions, together with other 1604 code, control the functions normally controlled by N and R.

With respect to time saved by not requiring LDR, it is relevant to note that the instruction has not been used in the sea surveillance problem to set up an "S" count. On the average, shift simulation in the 1604 should require 12.4 microseconds as opposed to the 8.1 microseconds required by LDR. The N and R parts of the LDR are simulated indirectly and need not be considered.

All input and output to and from the AMs are paralleled in A2 and GAP. A2 will have a slight timing advantage over GAP in that the LDR will not be required for the operations, and both A2 and GAP will average 7.1 microseconds per word transferred. The RCR of GAP is faster, however, than the similar operation in A2

APPENDIX A. PRELIMINARY PROGRAMMING MANUAL FOR THE GOODYEAR ASSOCIATIVE PROCESSOR

1. BASIC HYBRID SYSTEM (H_B)

a. External Functions

(1) Introduction

Certain functions generated by the external function instruction (EXF) in the 1604B can cause either a select or sense condition within the associative memory (AM). The EXF instruction contains a 12-bit operation code to be sent to the AM. This instruction causes the EXF counter in the 1604B to hold the operation code on the line for a sufficient period (8 usec) to ensure its proper sampling. When it receives its operation code, the AM can execute its portion of the instruction in less than 1.0 usec. The AM will respond to 16 different function codes. Each operation code generated by the 1604B and responded to by the AM is described briefly below.

(2) Clear All External Functions

This code resets all the sense and select flip-flops in the AM into their reset condition, and causes the AM to go inactive (HLT).

(3) Force

This causes the AM to resume operation. The AM forces its program counter to all ones $(77777)_8$. The program counter now will contain the next instruction address in the 1604B. This address should contain an instruction jump to the main program. The active line will be raised.

(4) Resume

This causes the AM to resume operation. The next AM instruction is taken from the address contained in the AM program counter. The active line will be raised.

(5) Interrupt on Parity Error

This operation code sets the parity interrupt flip-flop in the AM, and resets the parity flip-flop. The next parity error sets the parity flip-flop and turns on the interrupt line.

(6) Reset Parity Interrupt

This resets the parity interrupt and the parity flip-flop, and turns off the interrupt line.

(7) Interrupt on Overflow

An overflow occurs when more words are read into the AM than there are spaces to store them. This operation code resets the overflow interrupt flip-flop in the AM and resets the overflow flip-flop. When an overflow occurs, the overflow flip-flop is set and the interrupt line raised.

(8) Reset Overflow Interrupt

This operation code resets the overflow interrupt flip-flop and the overflow flip-flop, and lowers the interrupt line.

(9) Interrupt on Abnormal Condition

An abnormal condition can occur when an instruction or external function that is decoded by the AM is not a normal function. A negative address, an address greater than 2048, or when the AM is in an active condition and receives either a force or resume function, are examples of abnormal conditions. The interrupt on abnormal condition sets the abnormal interrupt flip-flop in the AM and resets the abnormal flip-flop. The interrupt line is raised when the error occurs.

(10) Reset Abnormal Interrupt

This code resets the abnormal interrupt flip-flop and the abnormal flip-flop, and lowers the interrupt line.

(11) Interrupt on AM Inactive

For a master clear, a clear all external functions, or a programmed halt

(HLT), the AM will go inactive. If the interrupt inactive is decoded before the AM goes inactive, the inactive interrupt flip-flop is set and the inactive flip-flop is reset. The interrupt line then is turned on when the AM goes inactive.

(12) Reset Inactive Interrupt

This code resets the inactive interrupt flip-flop and the inactive flip-flop, and lowers the interrupt line.

(13) Sense AM Active

This code causes the sense response line to be raised by the AM if it is active. If the AM is not active, the line will remain down.

(14) Sense Parity

If there has been a parity error detected since the previous clear or select operation, the AM will raise the sense response line. Otherwise, the line will remain down. Parity is checked each time an AM location is read.

(15) Sense Overflow

If any load operation (since the previous clear or select operation) has caused an overflow indication to occur, the sense response line will be raised by the AM. Otherwise, the line will remain down. Overflow occurs when more words are loaded into the AM than there are spaces available.

(16) Sense Abnormal Condition

If any external function or instruction is decoded as an abnormal condition after the previous clear or select operation has occurred, the sense response line will be raised by the AM. Otherwise, the line will remain down.

(17) Sense No Responder

This external function causes the sense response line to be raised at any time during the AM operation. Otherwise, the line will be down. This function is reset (no responders) at the start of each search instruction.

b. Format Description

(1) Introduction

Figure II-1(A) shows the AM instruction format for both the LDR and all other AM instructions, along with the number of bits decoded by the AM for each instruction, tag, and address. A description of all of the tags except those associated with LDR is given under (2) below. The LDR tags are described under d, below.

A suggested symbolic programming format (80 column) instruction card is shown in Figure II-1(B).

(2) Instruction Tag Description

(a) P - Portion

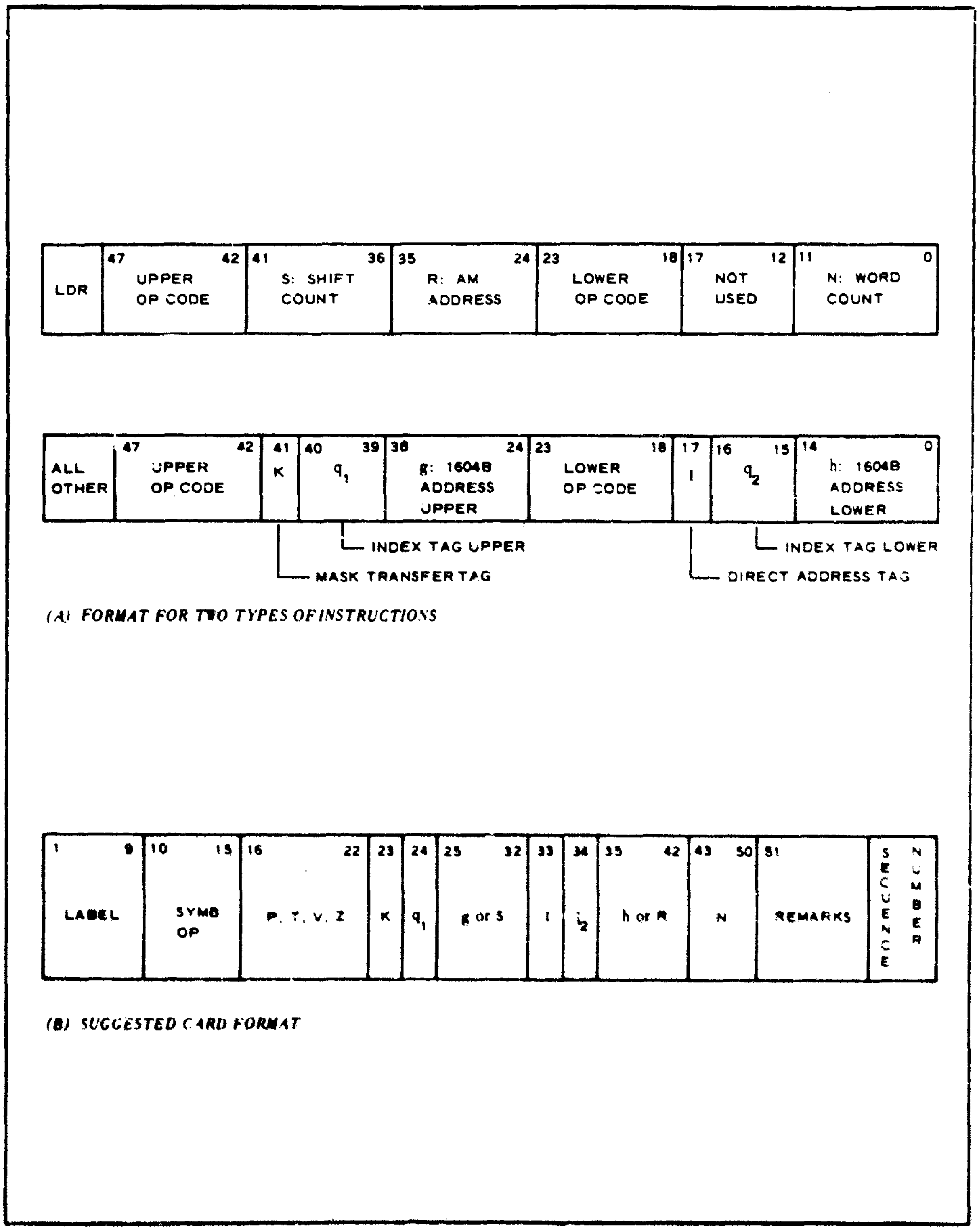
The P-portion tag allows the programmer to choose that half of the memory on which the instruction will operate. Either half or both halves may be chosen:

P	= 0	Whole memory
	= 1	Upper half of memory
	= 2	Lower half of memory
	= 3	No operation

(b) T - Transfer

This tag allows the programmer to advance the buffers before execution of the operation. Since the buffers normally hold the previous results and since the searches have a connective specified, this tag allows searches to be extended over adjacent locations.

Normal operation of the AM uses a T field of zero. When it is desired to connect a search of one part of a data word with another part of the same da a word stored in the next higher AM location, the second search instruction should contain a T field of one and a "connective" (see below) other than COPY.



	47 42	41 36	35 24	23 18	17 12	11 0
LDR	UPPER OP CODE	S: SHIFT COUNT	R: AM ADDRESS	LOWER OP CODE	NOT USED	N: WORD COUNT

	47 42	41	40 39	38 24	23 18	17	16 15	14 0
ALL OTHER	UPPER OP CODE	K	q_1	g: 1604B ADDRESS UPPER	LOWER OP CODE	I	q_2	h: 1604B ADDRESS LOWER

(A) FORMAT FOR TWO TYPES OF INSTRUCTIONS

1 9	10 15	16 22	23	24	25 32	33	34	35 42	43 50	51	
LABEL	SYMB OP	P, T, V, Z	K	q_1	g or S	I	q_2	h or R	N	REMARKS	SEQUENCE NUMBER

(B) SUGGESTED CARD FORMAT

Figu. II-1 - Associative Memory Instruction Format

A T = 1 tag causes the contents of the buffer to shift one position toward hig
core before execution of the instruction. The last bit of the D buffer shifts into the first bit of the E buffer. The last bit of the E buffer is shifted into the first bit of D.

T = 0 No advance of D and E buffers

= 1 Advance D and E buffers one place toward higher numbered location before operation

(c) V - Initialize

In normal operation (V = 0), the response store is initialized in one of several ways before commencing the execution of an instruction. For example, EMC initializes to all ONES, MMC to all ZEROS, and read and write operations involving the "responders" initialize the response store by copying first the D and then the E buffers into it. With a V = 1 tag, this initialization is inhibited and whatever may have been left in the response store by a prior operation is taken as the initial conditions. This can result in ANDing together successive EMC searches without disturbing the buffers or in ORing together successive MMC searches. It is expected that a V = 1 will be used only with operations on half of memory at a time (P = 1 or 2).

V = 0 Initialize response store in normal manner

1 Operate with preceding response store

(d) Z - Connective

This tag allows the programmer to choose the logical connective desired at the end of a search. This connective is used to specify how the previous results in the buffers will be affected by the new results in the response store. Normally, when a particular half of memory is searched the results are stored in the corresponding buffer so that the next search, when it uses the response store, will not destroy these results. However, the results of the last half-search performed are always found in the response store.

Given the V tag, it is possible to operate on the results in the response store only and a tag of Z equal to 3 may be used to guard the buffers. Note that if P = 0 the response store will end up with only the results of the search on upper (1024 - 2047) memory. The results from searching lower memory will be lost.

Z	= 0	Copy response store into D and E buffers
	= 1	AND connective of response store into buffers
	= 2	OR connective of response store into buffers
	= 3	No copy

(e) K - Mask Transfer

This tag allows the programmer to inhibit the fetch of a mask and thus use the same mask for several successive searches, saving the mask transfer time.

K	= 0	Specifies a mask transfer for the operation
	= 1	Inhibits a mask transfer for the operation (that is, old mask is used if operation requires a mask)

(f) I - Indirect Addressing

For certain instructions (JIH, JIL, JNR, JUC, LDI, SIX, and SPJ), indirect addressing can be specified by the "I" tag. An I = 0 specifies direct addressing; that is, H is used. An I = 1 implies "indirect addressing" and the contents of H (low order 18 bits of H) are used. A new I, q_2, and h are contained in these low-order 18 bits; hence, with continued I = 1, an infinite level of indirect addressing is possible.

I	= 0	Direct addressing (H is used as address)
	= 1	Indirect addressing (H is used to specify address)

(g) q_1 and q_2 - Index Tags

These two tags allow the programmer to modify the effective addrresses used by the AM in the usual fashion.

$q_1 = 0, G = g$
$= 1, G = g +$ (Index Register No. 1)
$= 2, G = g +$ (Index Register No. 2)
$= 3, G = g +$ (Index Register No. 3)

NOTE

(A) is read "the contents of A." (G is the effective 1604B address.)

$q_2 = 0, H = h$
$= 1, H = h +$ (Index Register No. 1)
$= 2, H = h +$ (Index Register No. 2)
$= 3, H = h +$ (Index Register No. 3)

NOTE

H is the effective 1604B address.

c. Instruction List

(1) Introduction

Eacn in-truction used by the AM in the basic hybrid system is listed below according to the function operations. Instructions that are prefixed by an asterisk (*) use information from the previous LDR instruction. Those instructions that are underlined are a direct result of this hybrid study.

(2) Load Operation

The load operation is:

LDR	Load AM registers (S, R, N)

(3) Input Operations

The input operations are:

1.	*WAL	Write into available locations (P, K, q_1, g, q_2, h)
2.	*WCO	Write conventionally (K, q_1, g, q_2, h)

3.	*WCR	Write constant into responders (P, T, V, K, q_1, g, q_2, h)
4.	*WIR	Write into responders (P, T, V, K, q_1, g, q_2, h)

(4) Output Operations

The output operations are:

1.	RAK	Read addresses of responders (P, T, V, q_2, h)
2.	RAF	Read address of first responder (P, T, V, q_2, h)
3.	*RCO	Read conventionally (q_2, h)
4.	*RCM	Read conventionally, monitored (q_2, h)
5.	RCR	Read count of responders (P, V, q_2, h)
6.	RCA	Read count of responders, accumulated (P, V, q_2, h)
7.	CRS	Count responders (P, V)
8.	CRA	Count responders, accumulated (P, V)
9.	*RDR	Read from responders (P, T, V, q_2, h)
10.	*RDF	Read first responders (P, T, V, q_2, h)

(5) Activate and Erase Operations

The activate and erase operations are:

1.	*ACO	Activate conventionally
2.	*ECO	Erase conventionally
3.	EMY	Erase memory (P)
4.	EAR	Erase all responders (P, T, V)
5.	EFR	Erase first responder (P, T, V)

(6) Index Operations

The index operations are:

1. DEI Decrement index (q_1, g)
2. ICI Increment index (q_1, g)
3. LDI Load index (q_1, I, q_2, h)
4. SIX Store index (q_1, I, q_2, h)

(7) Control Operations

The control operations are:

1. JUC Jump unconditionally (I, q_2, h)
2. JNR Jump on no response (I, q_2, h)
3. JIH Jump on index high (q_1, g, I. q_2, h)
4. JIL Jump on index low (q_1, g, I, q_2, h)
5. SPJ Store program counter and jump (q_1, g, I. q_2, h)
6. NOP No operation
7. HLT Halt all AM operations

(8) Search Operations

The search operations are:

1. *EMC Exact match of comparand (P, T, V, Z, K, q_1, g, q_2, h)
2. *MMC Mismatch of comparand (P, T, V, Z, K, q_1, g, q_2, h)
3. *LTC Less-than comparand (P, T, Z, K, q_1, g, q_2, h)
4. *GEC Greater-than or equal-to comparand (P. T, Z, K, q_1, g, q_2, h)

5.	*LEC	Less-than or equal-to comparand (P, T, Z, K, q_1, g, q_2, h)
6.	*GTC	Greater-than comparand (P, T, Z, K, q_1, g, q_2, h)
7.	*MIN	Minimum value (P, T, Z, K, q_1, g)
8.	*MAX	Maximum value (P, T, Z, K, q_1, g)
9.	*BLC	Between limiting comparands (P, T, Z, K, q_1, g, q_2, h)
10.	*NLC	Next lower-than comparand (P, T, Z, K, q_1, g, q_2, h)
11.	*NHC	Next higher-than comparand (P, T, Z, K, q_1, g, q_2, h)
12.	*CPX	Complex search (P, T, K, q_1, g, q_2, h)
13.	*SCF	Skip complex field
14.	END	End complex search operation
15.	COB	Complement buffers (P)

d. Instruction Description

(1) Introduction

Each instruction used by the AM hybrid system is described here and the tags associated with each instruction are shown. How the tags alter the instructions is explained.

(2) Load AM Registers Operation

(a) LDR Load AM Registers (S, R, N)

This instruction loads the AM shift-count register with the S field, loads the AM address register with the R field, and loads the AM word-count register with the N field. The next instruction is then fetched. The shift-count register is not changed by the shift execution; however, any following instruction utilizing the AM address register and/or the word count register destroys the contents of these registers.

Address modification does not apply to this instruction. All fields are considered as absolute numbers.

(b) S Field

The S field is bits 41 through 36 indicating the number of places the comparand register, CR, or output register, OR, will be shifted (shift is an end-around shift-right operation). The largest shift count permissible is 63_{10}, and the largest practical is 48_{10}, which returns the comparand register to its original state. A shift count of 47_{10} is equivalent to an end-around shift left on one position.

(c) R Field .

The R field is bits 35 through 24 and specifies an AM starting address. An R of 4095_{10} is the largest possible; an R of 2047_{10} is the largest usable with the 2048_{10} word AM.

(d) N Field

The N field is bits 11 through 0 and specifies the count to be supplied to the AM for certain input-output operations with $0 \leqq N \leqq 4095_{10}$. An $N \geqq 2048_{10}$ will cause overflow on AM input operations.

(3) Input-Output Operations

(a) Introduction

Load (write) operations transfer data words from the 1604B memory to the AM store. Unload (read) operations transfer data from the AM store to the 1604B memory. For all load operations, the busy bit is set to active status for each word written into the AM. Unload operations do not change the busy bit active status.

A mask word is required for load operations. The presence of a ZERO in the mask word (masking) inhibits writing in that bit position of memory. Each input-output instruction, except RAR and RCR, must be preceded by an LDR (load AM registers).

Address modification can occur in input-output operations; that is, q_1 and q_2 tags apply. Indirect addressing does not apply for input-output.

The input and output data words are shifted as specified by S except for RAR and RCR instructions. For load operations, the data word is transferred from 1604B memory into the comparand register and then shifted prior to writing into the AM store. For unload operations, the output register is filled with the contents of an AM cell and then shifted prior to transfer to 1604B memory.

In the following instructions, if V = 0, the term "responders" will be taken to mean those items whose buffer flip-flops have been set by a prior search. Should the V tag be ONE, then the last half of memory searched (see P tag description) will have determined the setting of the response store flip-flops. Items that correspond to those response store locations where the flip-flops are SET will be the "responders."

(b) Input Instructions

WAL - Write Into Available Locations (P, K, g_1, g, g_2, h). - The S and N fields of the LDR instruction must have been specified, with the R field not applicable. A search on busy bit locations for all inactive (available) AM locations is performed. The contents of 1604B address G are transferred to the AM mask register (unless inhibited by K = 1). Then a block of N words from sequential ascending 1604B memory locations, H, H + 1, H + 2, ... H + N - 1 is loaded into ascending available AM locations (not necessarily contiguous) starting with the first available (lowest numbered) location. The AM will generate an overflow interrupt if words still remain to be transferred after all available AM locations have been filled.

WCO - Write Conventionally (K, g_1, g, g_2 h). - The S, R, and N fields of the LDR instruction must have been specified. The contents of 1604B address G are placed in the AM mask register MR (unless inhibited by K = 1). Then a block of N words from sequential ascending 1604B-memory locations, H, H + 1, H + 2, . . . , H + N - 1, is loaded into sequential ascending AM locations starting at the AM address specified by R. The AM will generate an overflow interrupt if words still remain tobe transferred after AM location 2047_{10} has been filled.

WCR - Write Constant Into Responders ($P, T, V, K, q_1, g, q_2, h$). - The S field of the LDR instruction must have been specified, with R and N fields not applicable. This instruction transfers the mask word from 1604B memory location G (unless inhibited by K = 1), and the constant from 1604B location H is placed in the comparand register (CR) and then written into every responder.

WIR - Write Into Responders ($P, T, V, K, q_1, g, q_2, h$). - The S and N fields of the LDR instruction must have been specified, with the R field not applicable. The contents of 1604B address G are placed in the AM mask register (unless K = 1). Then a block of N words from sequential ascending 1604B-memory locations, H, H + 1, H + 2, . . . , H + n - 1, is loaded into successive responders, starting at the lowest location and proceeding to the highest. The AM will generate an overflow interrupt if words still remain to be transferred after all responders have been filled.

(c) Output Instructions

RAR - Read Addresses of Responders (P, T, V, q_2, h). - No LDR instruction is required for this operation. Each responder will have its address written into successive 1604B memory addresses starting with the address specified by H. For each responder, a 48-bit word is transferred to the 1604B, with bits 35 through 24 containing the 11-bit AM address All other bits are zero filled.

RAF - Read Address of First Responder (P, T, V, q_2, h). - This is the same as RAR above except that only the first (lowest address) responder has its address written into 1604B location H.

RCO - Read Conventionally (q_2, h). - The S, R, and N fields of the LDR instruction must have been specified. This operation unloads the contents of a block of N AM addresses, starting at the AM address specified by R, into sequential ascending 1604B memory locations starting at the 1604B address specified by H. The response store and buffer are not altered by this operation. Busy bit status will be ignored during unload.

RCM - Read Conventionally, Monitored (q_2, h). - This is identical to RCO except that every AM cell within the specified block to be unloaded that has a busy bit of ZERO (inactive), will have all ZEROS substituted as the output register contents for transfer to the 1604B memory. The contents of these AM cells that are inactive will not be altered within the AM itself.

RCR - Read Count of Responders (P, V, q_2, h). - No LDR instruction is required before this operation. This operation uses the response resolver and a resolver accumulator register to determine the number of responders. The count is transferred to the output register in bits 14 through 0 of a 48-bit word (all other bits ZERO filled) and transferred to 1604B memory at the address specified by H. The resolver accumulator register is cleared to ZERO before the counting operation begins.

RCA - Read Count of Responders, Accumulated (P, V, q_2, h). - This is the same as RCR except that the count of responses is added to the existing contents of the accumulator register.

CRS - Count Responder (P, V). - The manner of operation is identical to RCR except that the contents of the resolver accumulator register are not transferred to the 1604B.

CRA - Count Responders, Accumulated (P, V). - This is the same as RCA except that the contents of the resolver accumulator register are not transferred to the 1604B.

RDR - Read from Responders (P, T, V, q_2, h). - The S field of the LDR instruction must have been specified, with R and N fields not applicable. The contents of each responder are transmitted to successive 1604B locations beginning at H.

RDF - Read First Responder (P, T, V, q_2, h). - This is the same as RDR except that only the contents of the first (lowest address) responder will be transmitted to 1604B location H.

(d) **Activate and Erase Instructions.**

Activate and erase type instructions are provided to alter the status of the busy bit of AM cells. Although not an input operation (in the sense that data are physically transferred from 1604B to the AM), the effect on the busy bit is that of loading (writing) a ZERO or ONE into that bit position of AM cells. The other 48 bits of an AM cell are not altered by activate or erase operations. All the write operations of the previous section set the busy bits of the affected cells to ONE (active)

ACO - Activate Conventionally. - The R and N fields of the LDR instruction must have been specified, with the S field not applicable. This operation sets the busy bit of each AM cell to ONE (active) for a contiguous block of N cells (lower to higher), starting at the AM address specified by R. No other bits of the AM cell are altered.

ECO - Erase Conventionally. - This is identical to ACO except that the busy bit of each AM cell is set to ZERO (inactive).

EMY - Erase Memory (P). - No LDR instruction is required before this operation. The selected portion of the AM store (P = 0, 1, 2) will have the busy bit of every AM cell set to ZERO (inactive).

EAR - Erase Responders (P, T, V). - No LDR instruction is required before this operation. All AM cells containing respondents will have their busy bits set to ZERO (inactive).

EFR - Erase First Responder (P, T, V). - No LDR instruction is required before this operation. The cell containing the first (lowest address) responder will have its busy bit set to ZERO (inactive).

(4) **Index Operations**

(a) **Introduction**

There are three real index registers (1, 2, and 3) in the AM, and one virtual register (0) that contains all zeros. For all instructions except LDR, there are two address fields, g and h, and an index tag field associated with each address. For all instructions, the contents of the specified index register (0, 1, 2, or 3) are added to the contents of the instruction address field to obtain an "effective address" before a fetch from the 1604B is executed. The mask for search operations is obtained from 1604B location G, where G = g + (q_1) if K = ZERO. Otherwise, the previous mask will be retained. When a comparand is required, it is obtained from 1604B location H where H = h + (q_2). q_1 and q_2 may specify the same index register or they may be different.

Index registers are 15 bits long and operate with two's complement arithmetic, similar to those of the 1604B.

(b) Index Instructions

DEI - Decrement Index (q_1, g). - The index register specified by q_1 is decremented by the amount shown in field g. If q_1 = 0 specifying the virtual register, this instruction becomes a pass.

ICI - Increment Index (q_1, g). - The index register specified by q_1 is incremented by the amount shown in field g. If q_1 = 0, specifying the virtual register, this instruction becomes a pass.

LDI - Load Index (q_1, I, q_2, h). - The field q_1 specifies the index register to be loaded. If I = 0, the number in the H field is entered into q_1. If I = 1, the contents of the 18 right-most bits of the 1604B location H are loaded into the instruction register and reinterpreted as a new I, q_2, h. If q_1 = 0, this instruction becomes a pass.

SIX - Store Index (q_1, I, q_2, h). - The contents of the index register specified by q_1 are stored in the 15 right-most bits of 1604B location H when I = 0. If I = 1, the contents of 1604B location H (18 right-most bits) are loaded into the AM instruction register and reinterpreted as a new I, q_2, h. When q_1 = 0, this instruction stores zeros in the specified location.

(5) Control Operations

(a) Introduction

In normal operation, the program counter contains the address of the location in the 1604B at which the next AM instruction to be executed may be found. Its contents are incremented after the fetch and before the execution of each instruction. The operations discussed here can modify this normal sequence of instructions.

(b) Control Instructions

JUC - Jump Unconditionally (I, q_2, h). - If I = 0, this instruction causes the program counter to be loaded with the number specified by H. This causes a transfer of control to the sequence of AM instructions beginning at 1604B location H. If I = 1, the contents of the 18 right-most bits of 1604B location H are loaded into the instruction register and reinterpreted as I, q_2, h. This is, in effect, an "indirect jump" with an infinite level of indirectness.

JNR - Jump on No Responders (I, q_2, h). - This instruction examines the state of the report flip-flop (see section on searches). If this flip-flop is set, indicating "no" responders, this instruction becomes an unconditional jump to H or to the contents of the 18 right-most bits of H, depending on whether I = 0 or 1, respectively (see JUC above). If the report flip-flop is reset, indicating "some" responders, this instruction becomes a pass and the program counter is not modified.

JIH - Jump on Index High (q_1, g, I, q_2, h). - The contents of the index register specified by q_1 are compared with the number in the g field of this instruction. If the contents of the index register are greater than g, this becomes a jump to H or to the address specified by the 18 right-most bits of H, depending on the value of I (see JUC). If the contents of the index register are less than or equal to g, this instruction becomes a pass and the next instruction in sequence is executed.

JIL - Jump on Index Low (q_1, g, I, q_2, h). - The contents of the index register specified by q_1 are compared with the number in the g field of this instruction. If the contents of the index register are less than or equal to g, this becomes a jump to H or the contents of the 18 right-most bits of H, depending on the value of I (see JUC). If the contents of the index register are greater than g, this instruction becomes a pass.

SPJ - Store Program Counter and Jump (q_1, g, I, q_2, h). - The contents of the program counter (the location of the next instruction in sequence) are stored in the 15 right-most bits of 1604B location G. The remainder of the word at G is set to zero. The program counter is then loaded with H if I = 0. If I = 1, the contents of the 18 right-most bits of 1604B location H are reinterpreted as I, q_2, h.

NOP - No Operation. - This instruction does nothing and serves as a pass.

HLT - Halt All AM Operations. - This instruction terminates all operation in the AM, and "drops" the "AM active" line. The program counter will contain the address of the instruction immediately following this halt. If operation is reinitiated with a "resume" external function code by the 1604B, execution of instructions recommences at this address. If, however, a "force" external function code is given, the program counter is forced to all ONEs and takes the next AM instruction from 1604B location 77777_8.

(6) Search Operations

(a) Introduction

For all searches, the contents of the shift-count register, as determined by the last LDR instruction, will cause a right end-around shift of the comparand before the search is begun. The contents of the mask register are not shifted. Each search operation executed will reset the report flip-flop if any cells in the AM "respond" (that is, meet the search criterion). The report flip-flop will be set preceding any search.

The AM memory is divided into two halves: the lower, locations 0 through 1023, and the upper, locations 1024 through 2047. There is a bank of 1024 flip-flops called the D buffer associated with the lower and a similar bank called the E buffer associated with the upper. All searches (except for MAX and MIN) examine the lower half of memory and then the upper half. Temporary results of each half-search are stored in a third bank of 1024 flip-flops called the response store (RS). After each half-search is completed, the contents of the RS are gated into the appropriate buffer. There are four different modes of executing this gating; the mode that is employed is determined by the Z tag of the instruction. The chosen mode will be the same for both halves if both are searched. Not all the connectives can be used with all searches.

The results of the last half-search are left in the response store. In normal operation, all search operations first clear the RS to either all zeros or all ones depending on the algorithm employed. By writing a V tag of one (V = 1) with a search operation, this initial clear is inhibited and the "old" contents of the RS are used as the initial conditions.

NOTES

1. For all search operations any cells with a busy bit of zero are not eligible as responders. That is, each search operation includes an exact match operation on busy bit equal to one.

2. By "unmasked comparand" is meant the contents of the comparand register where bit positions for which the mask register contains zeros are not considered and the field defined by the ones in the mask is treated as if it were contiguous.

3. When K tag = 0, a mask is transferred; when K = 1, mask transfer is inhibited and old contents of mask register are used.

(b) Search Instructions

EMC - Exact Match of Comparand (P, T, V, Z, K, q_1, g, q_2, h). - Those cells in the AM containing words that match the comparand in every unmasked bit position will have their RS flip-flops set. Those cells containing words that mismatch the comparand in any unmasked position will have their RS flip-flops reset. V = 0 causes an initial set of the RS. V = 1 will AND the old contents to this search. Allowable Z connectives are 0, 1, 2, and 3.

MMC - Mismatch of Comparand (P, T, V, Z, K, q_1, g, q_2, h). - Those cells in the AM containing words that mismatch the comparand in any unmasked position will have their RS flip-flops set. Those cells containing words that match the comparand in every unmasked position will have their RS flip-flops reset. V = 0 causes an initial reset of the RS. V = 1 ORs the old contents of the RS with this search. Allowable Z connectives are 0, 1, 2, and 3.

LTC - Less-Than Comparand (P, T, Z, K, q_1, g, q_2, h). - Those cells in the AM containing words that are less than the unmasked comparand will have their RS flip-flops set. Those cells containing words that are greater than or equal to the unmasked comparand will have their RS flip-flops reset. Allowable Z connectives are 0, 1, 2, and 3.

GEC - Greater-Than or Equal-To Comparand (P, T, Z, K, q_1, g, q_2, h). - Those cells in the AM containing words that are greater than or equal to the unmasked comparand will have their RS flip-flops set. Those cells containing words that are less than the unmasked comparand will have their RS flip-flops reset. Allowable Z connectives are 0, 1, 2, and 3.

LEC - Less-Than or Equal-To Comparand (P, T, Z, K, q_1, g, q_2, h). - Those cells in the AM containing words that are less than or equal to the unmasked comparand will have their RS flip-flops set. Those cells containing words that are greater than the unmasked comparand will have their RS flip-flops reset. Allowable Z connectives are 0, 1, 2, and 3.

GTC - Greater-Than Comparand (P, T, Z, K, q_1, g, q_2, h). - Those cells in the AM containing words that are greater than the unmasked comparand will have RS flip-flops set. Those containing words that are less than or equal to the unmasked comparand will have their RS flip-flops reset. Allowable Z connectives are 0, 1, 2, and 3.

MIN - Minimum Value (P, T, Z, K, q_1, g). - The cell in the AM that contains the word that is least within the unmasked field will have its RS flip-flop set. All other RS flip-flops will be reset. The buffer is used for temporary storage, prohibiting the use of the OR and the NOCOPY connectives. No comparand is required. Allowable Z connectives are 0 and 1.

MAX - Maximum Value (P, T, Z, K, q_1, g). - The cell in the AM that contains the word that is greatest (within the unmasked field) will have its RS flip-flop set. All other RS flip-flops will be reset. The buffer is used for temporary storage, prohibiting the use of the OR and the NOCOPY connectives. No comparand is required. Allowable Z connectives are 0 and 1.

BLC - Between Limiting Comparands (P, T, Z, K, q_1, g, q_2, h). - Those cells containing words less than the unmasked comparand stored in 1604B location H and greater than the unmasked comparand stored in 1604B location H + 1 will have their RS flip-flops set. Cells containing words equal to or outside these limits will have their RS flip-flops reset. This instruction consists of an LTC search followed by a GTC search. The buffer is used for storage of the results of the LTC search, prohibiting the use of the OR and the NOCOPY connectives. The allowable Z connectives are 0 and 1.

NLC - Next Lower-Than Comparand (P, T, Z, K, q_1, g, q_2, h). - The cell in the AM which contains the word that is greatest within the unmasked field and still less than the unmasked comparand will have its RS flip-flop set. All other RS flip-flops will be reset. This instruction consists of an LTC search followed by a MAX search. The buffer is used for storage of the results of the LTC search prohibiting the use of the OR and the NOCOPY connectives. Allowable Z connectives are 0 and 1.

NHC - Next Higher-Than Comparand (P, T, Z, K, q_1, g, q_2, h). - The cell in the AM containing the word that is least within the unmasked field and still greater than the unmasked comparand will have its RS flip-flop set. All other RS flip-flops will be reset. This instruction consists of a GTC search followed by a MIN search. The buffer is used to store the results of the GTC search, prohibiting the use of the OR and the NOCOPY connectives. The allowable Z connectives are 0 and 1.

CPX - Complex Search (P, T, K, q_1, g, q_2, h). - This instruction permits the execution of up to eight connected searches on distinct, contiguous, non-overlapping fields of a word. A mask is fetched from 1604B location G. A "complex instruction" is fetched from 1604B location H. It consists of eight

6-bit fields, each of which specifies a search and a connective. A word is fetched from 1604B location H + 1 and is loaded into the field definition register. Each ONE in the field definition register marks the left edge of a field. Each field consists of the position marked by its initiating "ONE" and any ZERO positions that precede the next ONE. The left-most search is executed first on the first field, then the second search is executed on the second field and so forth. A comparand is fetched from 1604B location H + 2 and any other comparands required (by a BLC, for instance) come from H + 3, H + 4, etc. The searches that may be executed include all those above and the two additional codes that follow.

SCF - Skip Complex Field. - This code is used only within a complex instruction and indicates that no search is to be executed on the corresponding field. It serves as a NO-OP or pass.

END - End Complex Search. - This code is used within a complex instruction; it terminates the instruction when there are less than eight operations to be performed.

COB - Complement Buffers (P). - This operation is used to complement the D and/or E buffers, depending on P = 0, 1, or 2. The complement of a buffer is transferred into the response store, and a busy bit search is performed with AND connective. The response store is then transferred back to the buffer. With P = 0, both the D and the E buffers are complemented.

e. Instruction Timing

The setup and transfer timing for each instruction was obtained from the summation of the microtimes listed in Table IV-4 in Appendix IV (Page 140). The timing was analyzed on the basis of the work effort being accomplished on the actual hybrid associative memory hardware design. At the writing of this report, however, this design is not complete. For this reason, some of instruction time estimates will be less precise than others. For example, the various read/write instructions range from simple ones to those involving bit searching and resolving. The nominal times for these read/write instructions range from 4 to 5 usec per word. To simplify the timing analysis and to include any final design variations in the nominal times, a conservative 6 usec per word has been chosen for all read/write instructions.

TABLE II-1 - MAJOR SUBTIMES FOR ASSOCIATIVE MEMORY INSTRUCTIONS

Instruction	Subtime (μsec)	
	Minimum	Maximum
Instruction fetch (IF)	7.56	10.76
Decode instruction (DI)	0.04	0.04 (special cases)
Instruction housekeep (IH)	0.48	0.48 (normal cases)
Mask fetch (MF)	7.56	10.76
Mask housekeep (MH)	0.44	0.44
Comparand fetch (CF)	7.56	10.76
Search housekeep (SH)	0.40	0.80
AM memory - Memory access 1604B	8.753	11.953

The instruction timing breakdown follows.

1. LDR - Load AM registers (SRN)

	Minimum	Maximum
IF	7.56	10.76
DI	0.04	0.04
Gate to S, R, N registers	0.013	0.013
Set S, R, N registers	0.04	0.04
	7.653	10.853

t_{min} = 7.65 usec

t_{max} = 10.85 usec

2. WAL - Write into available locations (P, K, q_1, g, q_2, h)

	Minimum	Maximum
IF	7.56	10.76
IH	0.48	0.48
K = 1	8.04	11.24
MF if K = 0	7.56	10.76
MH	0.44	0 44
	16.04	22.44
CF (N words)	7.56N	10.76N
Write AM (N words)	6.00N	6.00N
	13.56N	16.76N

K = 1

t_{min} = 8.04 + 13.56N μsec

t_{max} = 11.24 + 16.76 N usec

K = 0

t_{min} = 16.04 + 13.56N μsec

t_{max} = 22.44 + 16.76N μsec

3. WCO - Write conventionally (K, q_1, g, q_2, h)

IF	7.56	10.76
IH	0.48	0.48
K = 1	8.04	11.24
K = 0 MF	7.56	10.76
MH	0.44	0.44
	16.04	22.44
CF/Write AM (N words)	13.56N	16.76N

K = 1

t_{min} = 8.04 + 13.56N μsec

t_{max} = 11.24 + 16.76N μsec

K = 0

t_{min} = 16.04 + 13.56N μsec

t_{max} = 22.44 + 16.76N μsec

4. WCR - Write constant into responders (P, T, V, h, q_1, g, q_2, h)

	Minimum	Maximum
K = 1		
IF + IH	8.04	11.24
CF	7.56	10.76
	15.60	22.00
K = 0		
MF + MH	8.00	11.20
	23.60	33.20
Write into AM (N responders)	6.00N	6.00N

K = 1

t_{min} = 15.6 + 6N μsec

t_{max} = 22.0 + 6N μsec

K = 0

t_{min} = 23.6 + 6N μsec

t_{max} = 33.2 + 6N μsec

5. WIR - Write into responders (P, T, V, K, q_1, g, q_2, h)

	Minimum	Maximum
K = 1		
IF + IH	8.04	11.24
K = 0		
MF + MH	8.00	11.20
	16.04	22.44
CF/Write AM (N words)	13.56N	16.76N

K = 1

$$t_{min} = 8.04 + 13.56N\ \mu sec$$

$$t_{max} = 11.24 + 16.76N\ \mu sec$$

K = 0

$$t_{min} = 16.04 + 13.56N\ \mu sec$$

$$t_{max} = 22.44 + 16.76N\ \mu sec$$

6. RAR - Read address of responders (P, T, V, q_2, h)

IF	7.56	10.76
DI	0.04	0.04
	7.60	10.80
Decode address of responders where N is the number of responders	0.013N	0.013N
AM address register set (N responders set)	0.04N	0.04N
Output register set to memory access 1604B (N words)	8.74N	11.94N
	8.793N	11.993N

$$t_{min} = 7.60 + 8.79N\ \mu sec$$

$$t_{max} = 10.80 + 11.99N\ \mu sec$$

7. RAF - Read address of first responder (P, T, V, q_2, h)

	Minimum	Maximum
IF + DI	7.60	10.80
AM address register set to memory access 1604B	8.78	11.98
	16.38	22.78
Decode address of responders	0.013	0.013

t_{min} = 16.39 μsec

t_{max} = 22.79 μsec

8. RCO - Read conventionally (q_2, h)

IF + DI	7.60	10.80
Read AM/access 1604B (N words)	14.753N	17.953N

t_{min} = 7.60 + 14.753N μsec

t_{max} = 10.80 + 17.953N μsec

9. RCM - read conventionally, monitored (q_2, h)

Same as RCO

t_{min} = 7.60 + 14.753N μsec

t_{max} = 10.80 + 17.953N μsec

10. RCR - Read count of responder (P, V, q_2, h)

IF + DI	7.60	10.80
Response count (single responder)	0.832[a]	0.832[a]
Response counter set	0.04	0.04

[a] 10 μsec (maximum) for 2048 responders.

	Minimum	Maximum
AM output register set to memory access 1604B	8.74	11.94
	17.212	23.612

t_{min} = 17.21 µsec

t_{max} = 23.61 µsec

11. RCA - Read count of responders accumulated (P, V, q_2, h)

	Minimum	Maximum
RCR values	17.21	23.61
Gate to adders	0.02	0.02
Adder propagation	0.14	0.14
	17.37	23.77

t_{min} = 17.37 µsec

t_{max} = 23.77 µsec

12. CRS - Count responders (P, V)

	Minimum	Maximum
IF + DI	7.60	10.80
Response count (single responder)	0.832[a]	0.832[a]
Response counter set	0.04	0.04
	8.472	11.672

[a] 10 µsec (maximum) for 2048 responders.

t_{min} = 8.472 µsec

t_{max} = 11.672 µsec

13. CRA - Count responders accumulated (P, V)

	Minimum	Maximum
CRS values	8.472	11.672
Gate to adders	0.02	0.02
Adder propagation	0.14	0.14
	8.632	11.832

t_{min} = 8.53 µsec

t_{max} = 11.83 µsec

14. RDR - Read from responders (P, T, V, q_2, h)

IF + DI	7.60	10.80
Read AM/access 1604B (N responders)	14.753	17.953

t_{min} = 7.60 + 14.753N µsec

t_{max} = 10.80 + 17.953N µsec

15. RDF - Read first responder (P, T, V, q_2, h)

IF + DI	7.60	10.80
Read AM/access 1604B	14.753	17.953
	22.353	28.753

t_{min} = 22.353 µsec

t_{max} = 28.75 µsec

16. ACO - Activate conventionally

	Minimum	Maximum
IF + DI	7.60	10.80
Set busy bit to zero (N words RD/WRT cycle)	6.00N	6.00N

$t_{min} = 7.60 + 6N\ \mu sec$

$t_{max} = 10.80 + 6N\ \mu sec$

17. ECO - Erase conventionally

Same as ACO

$t_{min} = 7.60 + 6N\ \mu sec$

$t_{max} = 10.80 + 6N\ \mu sec$

18. EMY - Erase memory (P)

IF + DI	7.60	10.80
P = 1 or 2, erase busy bit on low or high memory	5.00	5.00
	12.60	15.80
P = 0, erase busy bit on both low and high memories	5.00	5.00
	17.60	20.80

P = 1 or 2

$t_{min} = 12.60\ \mu sec$

$t_{max} = 15.80\ \mu sec$

P = 0

$t_{min} = 17.60\ \mu sec$

$t_{max} = 20.80\ \mu sec$

19. EAR - Erase responders (P, T, V)

	Minimum	Maximum
IF + DI	7.60	10.80
Set responders busy bit to zero (N responders)	6.00N	6.00N

$t_{min} = 7.60 + 6N\ \mu sec$

$t_{max} = 10.80 + 6N\ \mu sec$

20. DEI - Decrement index (q_1, g)

q_1 = 0, IF + DI	7.60	10.80
Sub propagation	0.14	0.14
Select index register q_1	0.013	0.013
Set index register q_1	0.04	0.04
	7.793	10.993

$q_1 = 0$

$$\left.\begin{array}{l} t_{min} = 7.60\ \mu sec \\ t_{max} = 10.80\ \mu sec \end{array}\right\}\text{pass or NOP}$$

$q_1 = 1, 2, 3$

$t_{min} = 7.79\ \mu sec$

$t_{max} = 10.99\ \mu sec$

21. ICI - Increment index (q_1, g)

Same as DEI except sub propagation equal adder propagation

$q_1 = 0$

$$\left.\begin{array}{l} t_{min} = 7.60\ \mu sec \\ t_{max} = 10.80\ \mu sec \end{array}\right\}\text{pass or NOP}$$

q_1 = 1, 2, 3

t_{min} = 7.79 µsec

t_{max} = 10.99 µsec

22. LDI - Load index (q, I, q_2, h)

	Minimum	Maximum
q_1 = 0		
IF + DI (pass or NOP when q_1 = 0)	7.60	10.80
I = 0, q_1 = 1, 2, or 3		
Gate to adders	0.02	0.02
Adder propagation	0.14	0 14
Select index register q_1	0.013	0.013
Set index register q_1	0.04	0.04
	7.813	11.013
I = 1		
IF + DI	7.64	10.8
Execute LDI	7.81	11.01
	15.45	21.81

q_1 = 0

t_{min} = 7.60 µsec

t_{max} = 10.80 µsec

I = 0, q_1 = 1, 2, 3

t_{min} = 7.81 µsec

t_{max} = 11.01 µsec

I = 1

t_{min} = 15.45 μsec

t_{max} = 21.81 μsec

23. SIX - Store index (q_1, I, q_2, h)

	Minimum	Maximum
I = 0		
Same as LDI for I = 0, q_1 = 1, 2, or 3	7.813	11.013
Access 1604B memory	8.753	11.953
	16.566	22.966

I = 0

t_{min} = 16.57 μsec

t_{max} = 22.97 μsec

I = 1

Same as LDI for I = 1

t_{min} = 15.45 μsec

t_{max} = 21.81 μsec

24. JUC - Jump unconditionally (I, q_2, h)

I = 0		
IF	7.56	10.76
IH	0.48	0.48
	8.04	12.24
I = 1		
IF + IH	8.04	12.24
IF	7.60	10.76
Gate to program counter	0.013	0.013
Set program counter	0.040	0.04
	15.693	23.053

I = 0

t_{min} = 8.04 µsec

t_{max} = 12.24 µsec

I = 1

t_{min} = 15.69 µsec

t_{max} = 23.05 µsec

25. JNR - Jump on no responders (I, q_2, h)

	Minimum	Maximum
Yes responders		
IF	7.56	10.76
DI	0.04	0.04
	7.60	10.80
No responders		
I = 0		
IF	7.56	10.76
IH	0.48	0.48
	8.04	11.24
I = 1		
IF + IH	8.04	11.24
Fetch address (H) from 1604B	7.60	10.76
Gate (H) to program counter	0.013	0.013
Set program counter in AM	0.040	0.040
	15.693	22.053

Yes responders

t_{min} = 7.60 µsec

t_{max} = 10.80 µsec

No responders (I = 0)

t_{min} = 8.04 μsec

t_{max} = 11.24 μsec

No responders (I = 1)

t_{min} = 15.69 μsec

t_{max} = 22.05 μsec

26. JIH - Jump on index high (q_1, g, I, q_2, h)

	Minimum	Maximum
Index register ≤ g, IF + DI	7.60	10.80
Test index	0.20	0.20
	7.80	11.0

IR>g

Same as JUC except add 0.2 for test index

$$\left.\begin{array}{l} I = 0 \\ \quad t_{min} = 8.24\ \mu sec \\ \quad t_{max} = 12.44\ \mu sec \\ I = 1 \\ \quad t_{min} = 15.89\ \mu sec \\ \quad t_{max} = 23.25\ \mu sec \end{array}\right\} IR>g$$

IR≤g

t_{min} = 7.80 μsec

t_{max} = 11.00 μsec

27. JIL - Jump on index low (q_1, g, I, q_2, h)

Same time as JIH

$$\left.\begin{array}{l} I = 0 \\ \quad t_{min} = 8.24\ \mu sec \\ \quad t_{max} = 12.44\ \mu sec \\ I = 1 \\ \quad t_{min} = 15.89\ \mu sec \\ \quad t_{max} = 23.25\ \mu sec \end{array}\right\} IR < g$$

$IR \geq g$

t_{min} = 7.80 µsec

t_{max} = 11.00 µsec

28. SPJ - Store program counter and jump (q, g, I, g_2, h)

	Minimum	Maximum
I = 0		
IF	7.56	10.76
IH	0.48	0.48
AM access of 1604B	8.753	11.953
	16.793	23.193
I = 1		
Same as I = 0	16.793	23.193
Fetch address (H) from 1604B	7.600	10.760
Gate (H) to program counter	0.013	0.013
Set program counter in AM	0.040	0.040
	24.446	34.006

I = 0

t_{min} = 16.79 µsec

t_{max} = 23.19 µsec

I = 1

t_{min} = 24.45 µsec

t_{max} = 34.01 µsec

29. NOP - No operation

	Minimum	Maximum
IF	7.56	10.76
DI	0.04	0.04
	7.60	10.80

t_{min} = 7.60 µsec

t_{max} = 1[illegible] µsec

30. HLT - Halt all AM operations

IF	7.56	10.76
DI	0.04	0.04
Turn off "AM active" line driver	0.05	0.05
Turn off "AM active" line terminators	0.05	0.05
Reset "AM active" FF in 1604B	0.20	0.20
	7.95	11.10

t_{min} = 7.95 µsec

t_{max} = 11.10 µsec

31. EMC - Exact match of comparand (P, T, V, Z, K, q, q_2, h)

	Minimum	Maximum
K = 1, P = 1 or 2		
IF	7.56	10.76
IH	0.48	0.48
CF	7.60	10.76
Search 49 bits (100 nsec/bit) + SH	5.30	5.30
	20.94	27.30
K = 0		
MF	7.56	10.76
MH	0.44	0.44
	28.94	38.60
P = 0		
Search additional 49 bits	5.3	5.3
	34.24	43.90

K = 1, P = 1 or 2

t_{min} = 20.94 usec

t_{max} = 27.3 μsec

K = 0, P = 1 or 2

t_{min} = 28.94 usec

t_{max} = 38.60 μsec

K = 0, P = 0

t_{min} = 34.24 usec

t_{max} = 43.90 usec

P = 3, time is same as NOP

32. MMC - Mismatch of comparand (P, T, V, Z, K, q_1, g, q_2, h)

Time same as for EMC

K = 1, P = 1 or 2

t_{min} = 20.94 usec

t_{max} = 27.30 usec

K = 0, P = 1 or 2

t_{min} = 28.94 usec

t_{max} = 38.60 usec

K = 0, P = 0

t_{min} = 34.24 usec

t_{max} = 43.90 usec

P = 3, time is the same as NOP

33. LTC - Less-than comparand (P, T, V, Z, K, q, g, q_2, h)

Same as for EMC

34. GEC - Greater than or equal to comparand (P, T, V, Z, K, q, g, q_2)

Same as for EMC

35. LEC - Less-than or equal-to comparand (P, T, Z, K, q, g, q_2, H)

Same as for EMC

36. GTC - Greater-than comparand (P, T, Z, K, q_1, g, q_2, h)

Same as for EMC

37. MIN - Minimum value (P, T, Z, K, q_1, g)

	Minimum	Maximum
P = 1 or 2, K = 1		
IF	7.56	10.76
IH	0.48	0.48
Search 49 bits at 300 nsec/bit + SH	0.40	14.80
	8.44	26.04
K = 0		
MF + MH	8.00	11.2
	16.44	37.24
P = 0		
49 bits are searched at 600 nsec/bit until half of the memory is eliminated from the search, then the search continues at 300 nsec/bit plus 100 nsec for house-keeping	0.3	14.70
	16.74	51.94

K = 1, P = 1 or 2

t_{min} = 8.44 μsec

t_{max} = 26.04 μsec

K = 0, P = 1 or 2

t_{min} = 16.44 μsec

t_{max} = 37.24 μsec

K = 0, P = 0

t_{min} = 16.74 μsec

t_{max} = 51.94 μsec

38. MAX - Maximum value (P, T, Z, k, q_1, g)

Same as MIN

39. BLC - Between limiting comparands (P, T, Z, K, q_1, g, q_2, h)

K = 1, P = 1 or 2	Minimum	Maximum
IF	7.56	10.76
IH	0.48	0.48
CF	7.56	10.76
Search (49 bits) at 100 nsec/bit + SH	5.30	5.30
CF	7.56	10.76
Search (49 bits) at 100 nsec/bit + SH	5.30	5.30
	33.76	43.36

P = 0 Search time + SH	10.6	10.6
	44.44	53.96
K = 0 MF + MH	8.00	11.2
	52.44	65.16

K = 1, P = 1 or 2

t_{min} = 33.84 μsec

t_{max} = 43.36 μsec

K = 1, P = 0

t_{min} = 44.44 μsec

t_{max} = 53.96 μsec

K = 0, P = 0

t_{min} = 52.44 μsec

t_{max} = 65.16 μsec

40. NLC - Next lower-than comparand (P, T, Z, K, q_1, h)

	Minimum	Maximum
IF	7.56	10.76
IH	0.48	0.48
CF	7.56	10.76
LTC search (49 bits)	5.30	5.30
And connective	0.20	0.20
MAX search (49 bits)	0.40	14.80
K = 1, P = 1 or 2	21.50	42.30

	Minimum	Maximum
K = 1, P = 1 or 2	21.50	42.30
LTC and MAX search (49 bits)	5.6	20.00
K = 1, P = 0	27.10	62.30
K = 0, P = 0		
MF	7.56	10.76
MH	0.44	0.44
	35.10	73.50

K = 1, P = 1 or 2

t_{min} = 21.5 μsec

t_{max} = 42.3 μsec

K = 1, P = 0

t_{min} = 27.1 μsec

t_{max} = 62.3 μsec

K = 0, P = 0

t_{min} = 35.10 μsec

t_{max} = 73.5 μsec

41. NHC - Next higher-than comparand (P, T, Z, K, q_1, g, q_2, h)

Time is same as for NLC, but substitute GTC for LTC and MIN for MAX

42. CPX - Complex search (P, T, K, q_1, g, q_2, h)

	Minimum	Maximum
IF	7.56	10.76
IF	7.56	10.76
IH	0.48	0.48
Field definition fetch	7.56	10.76
Field definition housekeeping	0.44	0.44
Mask fetch	7.56	10.76
Mask housekeeping	0.44	0.44
	31.60	44.40

NOTE

(1) This instruction can execute up to eight connected searches of any search combination. Thus, the time differs for each combination and depends on the search, the field definition, and the comparand fetches. As an example, for MIN or MAX there is no comparand fetch, but for eight BLC there are 16 comparand fetches.

(2) The number of fields (F) must equal the number of instructions (I). If F < I, then only F number of instructions will be executed and the remainder will be treated as an SCF (skip complex field).

43. SCF - Skip-complex field (used only with a complex instruction)

Decode instruction	0.04
Search N^1 bits for next field, 13 nsec/bit	$0.013N^1$

$t = 0.04 + 0.013N^1$ µsec

44. END - End complex search (used only with a complex instruction)

Decode instruction	0.04

t = 0.04 μsec

45. COB - Complement buffers (P)

	Minimum	Maximum
IF	7.560	10.760
IH	0.480	0.480
Complement buffer	0.100	0.100
Transfer into response store	0.013	0.013
Search busy bit	0.100	0.100
AND connective	0.200	0.200
Transfer into D or E buffer	0.013	0.013
P = 1 or 2	8.466	11.666
P = 0	0.426	0.426
	8.892	12.092

P = 1 or 2

t_{min} = 8.47 μsec

t_{max} = 11.67 μsec

P = 0

t_{min} = 8.89 μsec

t_{max} = 12.09 μsec

2. HYBRID SYSTEM WITH HIGH-SPEED STORE

a. Introduction

Timing in this section has been calculated on basis of all the instructions listed in the basic hybrid machine (H_B). Only a few of these instructions are used in the "optimized hybrid systems." It is conceivable that proble ns of a nature other than those explored under contract may use additional instructions; thus for completeness, all instructions are timed out.

b. External Functions

The external functions used in the hybrid system with high-speed store are the same as those for the basic hybrid system described previously in this appendix.

c. Format Description

The instruction format used in the hybrid system with high-speed store is the same as for the basic hybrid system previously described in this appendix.

d. Instruction List

The instruction list used in the hybrid system with high-speed store is the same as the one given in the basic hybrid system previously described in this appendix with the exception of one additional instruction. This new XMT (transmit) is an input-output instruction, and must be preceded by an LDA instruction. All the instructions, except XMT, have been described previously in this appendix.

For XMT (transmit) the S, R, and N fields of the LDR instruction must have been specified. The content of the H address is placed in the data address register (DAR) and the content of the G address is placed in a second data address register (*DAR). If G and H are greater than 4095, a block of N words is read from 1604B addresses G, G + 1, G + 2, etc., and written into 1604B addresses H, H + 1,

TABLE II-2 - ASSOCIATIVE MEMORY INSTRUCTION TIME USING A HIGH-SPEED STORE

Instruction	Time (μsec)	
	Minimum	Maximum
Instruction fetch (IF)	0.86	0.86
Decode instruction (DI)	0.04	0.04 (special cases)
Instruction housekeep (IH)	0.48	0.48 (normal cases)
Mask fetch (MF)	0.86	0.86
Mask housekeeping (MH)	0.44	0.44
Comparand fetch (CF)	7.56	10.76
Search housekeep (SH)	0.4	0.8
Write into memory AM	6.00	6.00
Read from memory AM	2.54	6.00
AM to access 1604 B	8.75	11.95
Write into HSS memory	1.00	1.00
Read from HSS memory	0.50	1.00

H + 2, etc. If G and H are less than 4096, a block of N words is read from the high-speed store addresses G, G + 1, G + 2, etc., and written into the high-speed store addresses H, H + 1, H + 2, etc. If G is greater than 4095 and H is less than 4096, a block of N words is read from 1604B addresses G, G + 1, G + 2, etc., and written into the high-speed store addresses H, H + 1, H + 2, etc. If G is less than 4096 and H is greater than 4095, a block of N words is read from the high-speed store addresses G, G + 1, G + 2, etc., and written into addresses H, H + 1, H + 2, etc.

e. Instruction Timing

Setup timing for each instruction was obtained from the summation of the microtimes listed in Tables IV-4 and IV-5 of Appendix IV of RADC-TR-65-445. **Minimum and maximum values (in microseconds) of** time are given for each instruction used by the AM. The major subtimes for each instruction are given in Table II-2.

The instruction timing follows.

1. LDR - Load AM registers

	Minimum	Maximum
IF	0.86	
DI	0.04	
Gate to S, R, N registers	0.013	
Set S, R, N registers	0.04	
	0.953	

$t = 0.95$ μsec

2. XMT - Transmit

a. Read from 1604B, WRT into HSS

IF (see time H_B)	7.56	10.76
IH	0.48	0.48
K = 1	8.04	11.24
MF if K = 0 (see time H_B)	7.56	10.76
MH	0.44	0.44
	16.04	22.44
CF N words (see time H_B)	7.56N	10.76N
WRT HSS N words	1.00N	1.00N
	8.56N	11.76N

K = 1

t_{min} = 8.04 + 8.56N μsec

t_{max} = 11.24 + 11.76N usec

K = 0

t_{min} = 16.04 + 8.56N μsec

t_{max} = 22.44 + 11.76N usec

b. Read from HSS; WRT into 1604B

	Minimum	Maximum
IF + ID (see time H_B)	7.60	10.80
Read from HSS (N words)	0.50N	1.00N
HSS to 1604B memory N words (see time H_B)	8.753N	11.953N
	9.253N	12.953N

t_{min} = 7.6 + 9.253N usec

t_{max} = 10.8 + 12.953N usec

3. WAL - Write into available locations

IF	0.86
IH	0.48
K = 1	1.34
MF if K = 0	0.86
MH	0.44
	2.64

	Minimum	Maximum
CR (N words)	7.56N	10.76N
Write into AM (N words)	6.00N	6.00N
	13.56N	16.76N

K = 1

t_{min} = 1.34 + 13.56N usec

t_{max} = 1.34 + 16.76N usec

K = 0

t_{min} = 2.64 + 13.56N usec

t_{max} = 2.64 + 16.76N usec

4. WCO - Write conventionally

IF	0.86	
IH	0.48	
K = 1	1.34	
K = 0 MF	0.86	
MH	0.44	
	2.64	
CF/Write AM (N words)	13.56N	16.76N

K = 1

$t_{min} = 1.34 + 13.56N$ μsec

$t_{max} = 1.34 + 16.76N$ μsec

K = 0

$t_{min} = 2.64 + 13.56N$ μsec

$t_{max} = 2.64 + 16.76N$ μsec

5. WCR - Write constant into responder

	Minimum	Maximum
K = 1		
IF + IH	1.34	1.34
CF	7.56	10.76
	8.90	12.10
K = 0		
MF + MH	1.30	1.30
	10.20	13.40
Write into AM	6N	

K = 1

$t_{min} = 8.9 + 6N$ μsec

$t_{max} = 12.1 + 6N$ μsec

K = 0

$t_{min} = 10.2 + 6N$ μsec

$t_{max} = 13.4 + 6N$ μsec

6. WIR - Write into responders

	Minimum	Maximum
K = 1		
IF + IH	1.34	
K = 0		
MF + MH	1.30	
	2.64	
CF/Write AM (N words)	13.56N	16.76N

K = 1

t_{min} = 1.34 + 13.56N usec

t_{max} = 1.34 + 16.76N usec

K = 0

t_{min} = 2.64 + 13.56N

t_{max} = 2.64 + 16.76N

7. RAR - Read address of responders

IF + DI	0.9
Decode address of responders where N is the number of responders	0.013N

AM address register set (N responders set)	0.04N	0.04N
Output register access 1604B(N)	8.74N	11.94N
	8.79N	11.99N

t_{min} = 0.9 + 8.79N µsec

t_{max} = 0.9 + 11.99N µsec

8. RAF - Read address of first responder same as RAR with N = 1

t_{min} = 9.69 µsec

t_{max} = 12.89 µsec

9. RCO - Read conventionally

	Minimum	Maximum
IF + DI	0.9	
Read from AM (N words)	6.00N	6.00N
AM access 1604B (N words)	8.753N	11.953N
	15.653N	17.953N

t_{min} = 0.9 + 15.653N µsec

t_{max} = 0.9 + 17.953N µsec

10. RCM - Read conventionally monitored

(Same as RCO)

11. RCR - Read count of responders

	Minimum	Maximum
IF + DI	0.9	0.9
Response count (single responder)	0.832[a]	0.832[a]
Response counter set	0.04	0.04
AM output register set to access 1604B	8.74	11.94
	10.512	13.712

t_{min} = 10.51 μsec

t_{max} = 13.71 μsec

12. RCA - Read count of responders accumulated

RCR values	10.51	13.71
Gate to adders	0.02	0.02
Adder propagation	0.14	0.14
	10.67	13.87

t_{min} = 10.67 μsec

t_{max} = 13.87 μsec

13. CRS - Count responders

IF + DI	0.9
Response count (single responder)	0.832[a]
Response counter set	0.4
	2.132

t = 2.13 μsec

[a] 10 μsec maximum for 2048 responders.

14. CRA - Count responders accumulated

	Minimum	Maximum
CRS values	2.132	
Gate to adders	0.02	
Adder propagation	0.14	
	2.292	

$t = 2.29\ \mu sec$

15. RDR - Read from responders

IF + DI	0.9	0.9
Read from AM (N responders)	6.00N	6.00N
AM to access 1604B (N responders)	8.75N	11.95N
	14.75N	17.95N

$t_{min} = 0.9 + 14.75N\ \mu sec$

$t_{max} = 0.9 + 17.95N\ usec$

16. RDF - Read first responder

(Same as RDR with N = 1)

$t_{min} = 15.65\ \mu sec$

$t_{max} = 18.85\ \mu sec$

17. ACO - Activate conventionally

	Minimum	Maximum
IF + DI	0.9	

		Minimum	Maximum
	Set busy bit to zero	6.0N	
	t = 0.9 + 6.0N μsec		
18.	ECO - Erase conditionally		
	(Same as ACO)		
19.	EMY - Erase memory		
	IF + DI	0.9	
	Erase busy bit	5.0	
	P = 1 or 2	5.9	
	Erase busy bit	5.0	
	P = 0	10.9	
	P = 1 or 2		
	t = 5.9 μsec		
	P = 0		
	t = 10.9 μsec		
20.	EAR - Erase responders		
	IF + DI	0.9	
	Set responders busy bit to zero (N responders)	6.0N	
	t = 0.9 + 6.0N μsec		
21.	EFR - Erase first responders		
	(Same as EAR with N = 1)		
	t = 6.9 μsec		

22. DEI - Decrement index

	Minimum	Maximum
IF + DI	0.90	
Subtract propagation	0.14	
Select index register q_1	0.013	
Set index register q_1	0.04	
	1.093	

$q_1 = 0$

$t = 0.90$ µsec

$q_1 = 1, 2, 3$

$t = 1.09$ µsec

23. ICI - Increment index

(Same as DEI except subtract propagation becomes adder propagation)

24. LDI - Load index

$q_1 = 0$

IF + DI	0.90

$q_1 = 1, 2, 3$

Gate to adders	0.02
Adder propagation	0.14
Select index register q_1	0.013
Set index register q_1	0.04
	1.113

	Minimum	Maximum
	1.113	
I = 1		
IF + DI	0.9	
	2.013	

q_1 = 0

t = 0.9 μsec

I = 0,

q_1 = 1, 2, or 3

t = 1.11 μsec

I = 1

t = 2.01 μsec

25. SIX - Store index

	Minimum	Maximum
I = 0		
Same as LDI with I = 0		
q_1 = 1, 2, or 3	1.113	1.113
Access 1604B memory	8.753	11.953
	9.866	13.066
I = 1		
Same as LDI with I = 1		
I = 0	2.013	

t_{min} = 9.87 μsec

t_{max} = 13.07 μsec

I = 1

t = 2.01 μsec

26. JUC - Jump unconditionally

	Minimum	Maximum
I = 0		
IF + IH	1.34	
I = 1	1.34	
IF	0.86	
Gate to program counter	0.013	
Set program counter in AM	0.04	
	2.253	

I = 0

t = 1.34 μsec

I = 1

t = 2.25 μsec

27. JNR - Jump on no responders

Yes responders	
IF + DI	0.9
No responders	
I = 0	
IF + IH	1.34

	Minimum	Maximum
No responders		
I = 1		
IF + IH	1. 34	
Fetch address (H)	1. 3	
Gate (H) to program counter	0. 013	
Set program counter in AM	0. 04	
	2. 693	
Yes responders		
t = 0.9 usec		
No responders		
I = 0		
t = 1. 34 usec		
No responders		
I = 1		
t = 2. 69 usec		

28. JIH - Jump on index high

IR ≤ g

IF + DI	0.9	
Test index	0.2	
	1.1	

IR > g same as JUC except add 0.2 for test index

$IR > g$

I = 0

t = 1.54 μsec

I = 1

t = 2.45 μsec

$IR \leq g$

t = 1.10 μsec

29. JIL - Jump on index low

Same as JIH

$IR < g$

I = 0

t = 1.54 μsec

I = 1

t = 2.45 μsec

$IR \geq g$

t = 1.10 μsec

30. SPJ - Store program counter and jump

	Minimum	Maximum
I = 0		
IF + IH	1.34	1.34
AM to Access 1604B	8.753	11.953
	10.093	13.293
I = 1		
Same as I = 0	10.093	13.293

	Minimum	Maximum
Fetch address (H)	1.3	1.3
Gate (H) to program counter	0.013	0.013
Set program counter in AM	0.040	0.040
	11.446	14.646

I = 0

t_{min} = 10.09 µsec

t_{max} = 13.29 µsec

I = 1

t_{min} = 11.45 usec

t_{max} = 14.65 usec

31. NOP - No operation

IF + DI	0.9

t = 0.9 µsec

32. HLT - Halt all AM operations

IF + DI	0.9
Turn off "AM Active" line drivers	0.05
Turn off "AM Active" line terminators	0.05
Reset AM active FF in 1604B	0.20
	1.20

t = 1.2 usec

33. EMC - Exact match of comparand

	Minimum	Maximum
K = 1		
P = 1 or 2		
IF + IH	1.34	1.34
CF	7.56	10.76
Search 49 bits (100 nsec/bit) + SH	5.3	5.3
	14.20	17.40
K = 0		
MF + MH	1.30	1.30
	15.50	18.70
P = 0		
Search additional 49 bits + SH	5.3	5.3
	20.80	24.00

K = 1, P = 1 or 2

t_{min} = 14.2 μsec

t_{max} = 17.4 μsec

K = 1, P = 0

t_{min} = 19.5 μsec

t_{max} = 22.7 μsec

K = 0, P = 1 or 2

t_{min} = 15.5 μsec

t_{max} = 18.7 μsec

K = 0, P = 0

t_{min} = 20.8 μsec

t_{max} = 24.0 μsec

P = 3

time is same as NOP

34. MMC - Mismatch of comparand

(Time is same as EMC)

35. LTC - Less than comparand

(Time is same as EMC)

36. GEC - Greater than or equal to comparand

(Time as same as EMC)

37. LEC - Less than or equal to comparand

(Time is same as EMC)

38. GTC - Greater than comparand

(Time is same as EMC)

39. MIN - Minimum valu

P = 1 or 2

K = 1	Minimum	Maximum
IF + IH	1.34	1.34
Search 49 bits at 300 nsec/bit + SH	0.4	14.8
	1.74	16.14

	Minimum	Maximum
K = 0		
MF + MH	1.30	1.30
	3.04	17.44
P = 0 (49 bits are searched at 600 nsec/bit until half of the memory is eliminated from the search, then the search continues at 300 nsec/bit plus 100 nsec for housekeeping)	0.3	14.7
	3.34	32.14

K = 1, P = 1 or 2

t_{min} = 1.74 usec

t_{max} = 16.14 usec

K = 0, P = 1 or 2

t_{min} = 3.04 usec

t_{max} = 17.44 usec

K = 0, P = 0

t_{min} = 3.34 usec

t_{max} = 32.14 usec

40. MAX - Maximum value

(Time is same as for MIN)

41. BLC - Between limiting comparands

K = 1, P = 1 or 2

	Minimum	Maximum
IF + IH	1.34	1.34
CF	7.56	10.76
Search 49 bits at 100 nsec/bit + SH	5.3	5.3
CF	7.56	7.56
Search 49 bits at 100 nsec/bit + SH	5.3	5.3
	27.06	30.26
P = 0 two 49-bit searches + SH	10.6	10.6
	37.66	40.86
K = 0 (MF + MH)	1.30	1.30
	38.96	42.16

K = 1, P = 1 or 2

t_{min} = 27.06 μsec

t_{max} = 30.26 μsec

K = 1, P = 0

t_{min} = 37.66 μsec

t_{max} = 40.86 μsec

K = 0, P = 0

t_{min} = 38.96 μsec

t_{max} = 42.16 μsec

42. NLC - Next lower than comparand

IF + IH	1.34	1.34
CF	7.56	10.76
	8.90	12.10

	Minimum	Maximum
LTC - Search (49 bits)	5.3	5.3
AND connective	0.2	0.2
MAX search (49 bits)	0.4	14.8
K = 1, P = 1 or 2	14.8	32.4
LTC and MAX search (49 bits)	5.6	20.0
K = 1, P = 0	20.4	52.4
MF + MH	1.3	1.3
K = 0, P = 0	21.7	53.7

K = 1, P = 1 or 2

t_{min} = 14.8 μsec

t_{max} = 32.4 μsec

K = 1, P = 0

t_{min} = 20.4 μsec

t_{max} = 52.4 μsec

K = 0, P = 0

t_{min} = 21.7 μsec

t_{max} = 53.7 μsec

43. NHC - Next higher than comparand

Time is same as for NLC, but GTC replaces LTC and MIN replaces MAX

44. CPX - Complex search

	Minimum	Maximum
IF + IH	1.34	
IF	0.86	
Field definition fetch	0.86	
Field definition house-keeping	0.44	
Mask fetch	0.86	
Mask housekeeping	0.44	
	4.80	
Comparand fetch from 1604B	7.56	10.76

NOTE

Complete times for this instruction will differ depending on the combination of searches to be performed and the type of field definitions

45. SCF - Ship complex field (used only with a complex instruction)

Decode instruction	0.04
Search N^1 bits for next instruction field (13 nsec/-bit)	$0.013N^1$

$$t = 0.04 + 0.013N^1 \ \mu sec$$

46. END - End complex search (used only with a complex instruction)

Decode instruction	0.04

$$t = 0.04 \ \mu sec$$

47. COB - Complement buffers

IF + IH	1.340

	Minimum	
Complement buffers	0.100	
Transfer into response store	0.013	
Search busy bit	0.100	*
AND connective	0.200	
Transfer into D or E buffers	0.013	
P = 1 or 2	1.766	
*	0.426	
P = 0	2.192	

P = 1 or 2

t = 1.77 usec

P = 0

t = 2.19 usec

3. OPTIMIZED HYBRID SYSTEM WITH HSS (H_1)

a. External Functions

The external functions used in H_1 are the same as those used in H_B as described in paragraph 1, a of this appendix.

b. Format Description

The instruction format used in H_1 is the same as described in H_B, paragraph 1, b of this appendix.

c. Instruction List

Each instruction used by H_1 is listed below according to the function operation. Instructions that are prefixed by an asterisk (*) use information from the previous LDR instruction.

1. Load Operation

LDR	Load AM Register (S, R, N)

2. Input Operations

*WAL	Write into Available Locations (P, K, q_1, g, q_2, h)
*WCO	Write Conventionally (K, q_1, g, q_2, h)
*WCR	Write Constant into Responders (P, T, V, K, q_1, g, q_2, h)
*XMT	Transmit (G > 4095, H < 4096)

3. Output Operations

*RDR	Read from Responders (P, T, V, q_2, h)
*XMT	Transmit (G < 4096, H > 4096)

4. Erase Operations

*ECO	Erase Conventionally
EMY	Erase Memory (P)

5. Control Operations

JUC	Jump Unconditionally (I, q_2, h)
JNR	Jump on No Response (I, q_2, h)
NOP	No Operation
HLT	Halt all H_1 Operations

6. Search Operation

*EMC	Exact Match of Comparand (P, T, V, Z, K, q_1, g)

d. Instruction Description

Description of all H_1 instructions are given in **paragraphs 1, d and 2, d of this appendix.**

e. Instruction Timing

Times for all H_1 instructions are given in **paragraph 2, e of this appendix.**

4. OPTIMIZED HYBRID SYSTEM WITH HSS (H_{21})

a. External Functions

The external functions used in H_{21} are the same as those used in H_B described **in paragraph 1, g.**

b. Format Description

The instruction format is the same as described in System H_B, Item 1, b.

c. Instruction List

Each instruction used in system H_{21} is listed below according to the function operation. Instructions that are prefixed by an asterisk (*) use information from the previous LDR instruction.

1. Load Operation

LDR	Load AM registers (S, R, N)

2. Input Operations

*WCO	Write Conventionally (K, q_1, g, q_2, h)
*XMT	Transmit (G < 4095, H > 4096)

3. Output Operations

RCR	Read Count of Responders (P, V, q_2, h)
RCA	Read Count of Responders Accumulated (P, V, q_2, h)

	*RDR	Read From Responders (P, T, V, q_2, h)
	*XMT	Transmit (G < 4096, H > 4096)
4.	Index Operation	
	LDI	Load Index (q_1, I, q_2, h)
5.	Control Operations	
	JUC	Jump Unconditionally (I, q_2, h)
	NOP	No Operation
	HLT	Halt all H_{21} Operations
f.	Search Operation	
	*EMC	Exact Match of Comparand (P, T, V, Z, K, q_1, g)

d. Instruction Description

Description of all H_{21} nstructions are given in paragraphs 1, d and 2, d.

e. Instruction Timing

Time for all H_{21} instructions are given in paragraph 2, e.

5. OPTIMIZED HYBRID SYSTEM WITH HSS (H_{22})

a. External Functions

The external functions are described in paragraph 1, a.

b. Format Description

The instruction format is described in paragraph 1, b.

c. Instruction List

Each instruction used in system H_{22} is listed below according to the function operation. Instructions that are prefixed by an asterisk (*) use information from the previous LDR instruction.

1. Load Operation

LDR	Load AM Registers (S, R, N)

2. Input Operations

*WCO	Write Conventionally (K, q_1, g, q_2, h)
*XMT	Transmit (G > 4095, H < 4096)

3. Output Operations

RCR	Read Count of Responders (P, V, q_2, h)
*RDR	Read From Responders (P, T, V, q_2, h)
*XMT	Transmit (G < 4096,. H > 4095)

4. Control Operations

JUC	Jump Unconditionally (I, q_2, h)
NOP	No Operation
HLT	Halt all H_{22} Operations

5. Search Operation

*EMC	Exact Match of Comparand (P, T, V, Z, K, q_1, g)

d. Instruction Description

Description of H_{22} instructions are given in **paragraphs 1, d and 2, d.**

e. Instruction Timing

Times for all H_{22} instructions are given in **paragraph 2, e.**

6. OPTIMIZED HYBRID SYSTEM (H_{23})

a. General

Only a few basic instructions are needed for the H_{23} optimized hybrid machine configuration. The decoding of these instructions by the H_{23} causes the system to jump to a wired-in subprogram (H_{23} algorithm). For this reason, all instructions generated by the 1604B are treated as "external functions."

b. External Function List

Each external function used by the H_{23} is listed below.

1.	FORCE	Load delta store register
2.	LOAD	Load AM
3.	INT ZERO RSP	Interrupt on zero response
4.	INT MTO RSP	Interrupt on more than one response
5.	INT AB ERR	Interrupt on abnormal error
6.	INT PA ERR	Interrupt on parity error
7.	SEN CHAN ACT	Sense associative memory active
8.	SEN ZERO RSP	Sense for zero responders
9.	SEN PA ERR	Sense for parity error
10.	SEN MTO RSP	Sense for more than one responses
11.	SEN AB ERR	Sense for abnormal error
12.	SCA-1	Spelling corrector algorithm, single precision
13.	SCA-2	Spelling corrector algorithm, double precision

c. External Function Description

(1) FORCE - Load Delta Store Register

The lower half of the 1604B address $(77777)_8$ is stored in an address delta register. When it receives a force, the H_{23} fetches the delta address from the 1604B, stores it in the delta store register, and goes inactive.

(2) LOAD - Load AM

The contents of the 1604B address $(77777)_8$ contains the starting address in the 1604B and the H_{23}, and the number of words to be transferred from the 1604B to the H_{23}. When the H_{23} receives a load, it fetches the contents of the 1604B address $(77777)_8$ and stores the 1604B starting address in the data address register, the H_{23} starting address in the AM address register, and the word count in the word count register. The H_{23} then is block-loaded until the word count goes to zero, at which time it goes inactive.

(3) INT ZERO RSP - Interrupt on Zero Response

The interrupt on zero response sets the zero response interrupt flip-flop in the AM and resets the zero response flip-flop. The interrupt line is raised when there is no response to a search.

(4) INT MTO RSP - Interrupt on More than One Response

The interrupt on more than one response sets the MTO response interrupt flip-flop in the AM and resets the MTO response flip-flop. The interrupt line is raised when there is more than one response to a search.

(5) INT AB ERR - Interrupt on Abnormal Error

An abnormal condition occurs when a function that is not normal is decoded by the H_{23}. The interrupt on abnormal condition causes the abnormal interrupt flip-flop to be set and the abnormal flip-flop to be reset. The interrupt line is raised when an abnormal error occurs.

(6) INT PA ERR - Interrupt on Parity Error

This operation sets the parity interrupt flip-flop and resets the parity flip-flop. The next parity error sets the parity flip-flop and turns on the interrupt line.

(7) SEN CHAN ACT - Sense Associative Memory Active

This code causes the sense response line to be raised by the AM if it is active. If the AM is not active, then the line will remain down.

(8) SEN ZERO RSP - Sense for Zero Responders

If there is no response after a search operation, the AM will raise the sense response line; otherwise, the line will remain down.

(9) SEN PA ERR - Sense for Parity Error

If there has been a parity error detected since the previous select operation, the AM will raise the sense response line; otherwise. the line will remain down.

(10) SEN MTO RSP - Sense for More than One Responders

If there are more than one responders after a search operation, the AM will raise the sense response line; otherwise, the line will remain down.

(11) SEN AB ERR - Sense for Abnormal Error

If any function is decoded as an abnormal condition (since the previous select operation has occurred), the sense response line will be raised by the AM; otherwise, the line will remain down.

(12) SCA-1 - Single-Precision Spelling Correction Algorithm

When the SCA-1 instruction is decoded by the AM, the function ready line is raised. This initiates the spelling corrector algorithm. The single-precision algorithm search uses an 8-character or 48-bit word. The word to be corrected is loaded into the comparand register. This word can have no errors or a single error, where adjacent transponse characters are considered to be a single error. A search is made to determine the correct word. If at the end of the search there is no response, the interrupt zero response line is raised and the AM goes inactive. If there is one response and it is a correct word, the AM goes inactive. If there is one response and it is not the correct word, the AM sends the correct word to the delta plus one address in the 1604B and then goes inactive. If there is more than one response, the interrupt on more than one response line is raised, the number of responses is sent to the 1604B delta address, and the corrected words are sent to the 1604B addresses delta plus one, delta plus two, etc.

(13) SCA-2 - Double-Precision Spelling Corrector Algorithm

This instruction performs the same function as SCA-1; however, it can operate on a 16-character or 96-bit word. Thus it takes two 1604B words to equal one word for this instruction.

d. External Function Timing

The AM can execute an external function instruction in less than 1.0 usec with the exception of the three functions timed out below (FORCE, LOAD, and SCA). Minimum and maximum values (in microseconds of time are given for these instruction along with the major subtimes.

	Minimum	Maximum
1. FORCE - Load delta store register		
Load delta	7.56	10.76
Go inactive	0.10	0.10
	7.66	10.86

$t_{min} = 7.66\ \mu sec$

$t_{max} = 10.86\ \mu sec$

	Minimum	Maximum
2. LOAD - Load AM		
Load AM registers	7.56	10.76
Load AM with N words	21.12N	27.52N
Go inactive	0.1	0.1

$t_{min} = 7.66 + 21.12N\ \mu sec$

$t_{max} = 10.86 + 27.52N\ \mu sec$

3. SCA - Spelling corrector algorithm

This instruction is subdivided into two classes: SCA-1, spelling corrector algorithm single precision, and SCA-2, double precision.

	Minimum	Maximum
Subfunction time		
Load comparand SCA-1	7.56	10.76
Load comparand SCA-2	15.12	21.52
EMC $2 \leq K \leq 16$ (K = number of characters)	1.2	9.6
α, β, γ search $2 \leq K \leq 16$	3.0	27.6
α, β, γ zero response	0.35	0.35
α, β, γ one response	11.3	17.9
More than one response (R = number of responders)	11.3R	17.9R
Transfer accumulator count	6.8	9.8
SCA-1 for the correct word		
Load comparand SCA-1	7.56	10.76
EMC $2 \leq K \leq 8$	1.20	4.8
	8.76	15.56

$t_{min} = 8.76$ μsec

$t_{max} = 15.56$ μsec

	Minimum	Maximum
SCA-1 more than one response		
SCA-1 correct word	8.76	15.56
α, β, γ search $2 \leq K \leq 8$	3.00	13.8
Transfer accumulator count	6.8	9.8
	18.56	39.16
More than one response	11.3R	17.9R

$t_{min} = 18.56 + 11.3R$ μsec

$t_{max} = 37.96 + 17.9R$ μsec

	Minimum	Maximum
SCA-1 zero response		
SCA-1 correct word	8.76	15.56
α, β, γ search $2 \leq K \leq 8$	3.00	13.8
α, β, γ zero response	0.35	0.35
	12.11	29.71
$t_{min} = 12.11\ \mu sec$		
$t_{max} = 28.51\ \mu sec$		
SCA-1 one response		
SCA-1 correct word	8.76	15.56
α, β, γ search $2 \leq K \leq 8$	3.00	13.80
α, β, γ one response	11.3	17.9
	23.06	47.26
$t_{min} = 23.06$ usec		
$t_{max} = 47.26$ usec		
SCA-2 correct word		
Load comparand SCA-2	15.12	21.52
EMC $2 \leq K \leq 16$	1.2	9.6
	16.32	31.12
$t_{min} - 16.32$ usec		
$t_{max} = 31.12$ usec		
SCA-2 more than one word		
SCA-2 correct word	16.32	31.12
α, β, γ search $2 \leq K \leq 16$	3.0	27.6
Transfer accumulator count	6.8	9.8
	26.12	68.52
More than one response	11.3R	17.9R

$$t_{min} = 26.12 + 11.3R \ \mu sec$$

$$t_{max} = 68.52 + 17.9R \ \mu sec$$

	Minimum	Maximum
SCA-2 zero response		
SCA-2 correct word	16.32	31.12
α, β, γ search $2 \leq K \leq 16$	3.0	27.6
α, β, γ zero response	0.35	0.35
	19.67	59.07

$$t_{min} = 19.67 \ \mu sec$$

$$t_{max} = 59.07 \ \mu sec$$

	Minimum	Maximum
SCA-2 one response		
SCA-2 correct word	16.32	31.12
α, β, γ search $2 \leq K \leq 16$	3.0	27.6
α, β, γ one response	11.3	17.9
	30.62	76.62

$$t_{min} = 30.62 \ \mu sec$$

$$t_{max} = 76.62 \ \mu sec$$

APPENDIX B. DETAILED TIMING EQUATIONS

B.1 RESOLVER TIMING OF RANDOM RESPONSES

The problem may be expressed as follows. There are 32 rows of 32 bits each. The rows are examined one by one in fixed order until, for the first time, a row is found which contains a ONE. The examination takes one unit of time in either event. Having found such a row, the bits in it are examined, one at a time, until a ONE is found. In all, how many steps are required, on the average ?

The answer depends, of course, on what kind of data may be found in the register. Two alternative reasonable assumptions are:

(1) There is a fixed, known probability, k, that any given bit is a ONE.

(2) There is a fixed, known number, k, of ONEs, scattered at random throughout the register.

For purposes of analysis, the first assumption is somewhat simpler which is adopted here. It is likely that comparable estimates would result in the second case, using p = k/1024.

The expected or average number of steps is the sum for all possible locations of the first (or next) ONE of the number of steps required to reach the ONE multipled by the probability that the first ONE appears in the selected position. Thus, if q = 1-p, the contribution from a ONE in the first position of the first row is 2p. The second position in the first row contributes 3pq; the third position adds 4pq^2, etc.

The exact formula for E(n), the expected number of steps required, is

$$\begin{aligned} E(n) = 2p + 3pq + 4pq^{2} + \ldots + 32pq^{30} + 33pq^{31} & \\ + 3pq^{32} + 4pq^{33} + \ldots + 32pq^{61} + 33pq^{62} + 34pq^{63} & \\ + \ldots\ldots \qquad + 34pq^{993} + \ldots & \qquad (1) \\ + 64pq^{1023} & \end{aligned}$$

The first row of this formula has already been justified. The probability that the first ONE is in the first position of the second row is the probability that no ONE is in the first row (q^{32}) times the probability that the first bit of the second row is a ONE (p). In this case, however, only three steps are needed: check the first row, check the second row, check the first bit position of the second row. Formula (1) should now be evident. It can be condensed as follows:

$$\begin{aligned} E(n) = {} & 2p + 3pq(1+q^{31}) + 4pq^{2}(1+q^{31}+q^{62}) + \ldots \\ & + 33pq^{31}(1+q^{31} + \ldots + q^{961}) \\ & + 34pq^{32}(q^{31} + \ldots + q^{961}) \\ & + 35pq^{33}(q^{62} + \ldots + q^{961}) \\ & + \ldots + 64pq^{62}(q^{961}) \end{aligned} \tag{2}$$

The question is how to evaluate (2) for selected values of p. If q^{31} is much less than 1, we may replace all factors $(1 + q^{31} + \ldots)$ by 1, with little error. In this case, (2) becomes,

$$\begin{aligned} E(n) & \cong 2p + 3pq + 4pq^{2} + \ldots + 33pq^{31} \\ & = \frac{p}{q}(2q + 3q^{2} + \ldots + 33q^{32}) \\ & = \frac{p}{q}\left(-1 + \frac{d}{dq} \sum_{j=0}^{33} q^{j}\right) \\ & = \frac{p}{q}\left(-1 + \frac{d}{dq}\,\frac{1-q^{34}}{1-q}\right) \\ & = \frac{p}{q}\left(-1 + \frac{(1-q)(-34q^{33}) + (1-q^{34})}{(1-q)^{2}}\right) \end{aligned} \tag{3}$$

$$E(n) \cong \frac{p}{q}\left(-1 + \frac{1}{(1-q)^{2}}\right) = \frac{p}{q} + \frac{1}{pq} = \frac{1-p^{2}}{pq} = \frac{1+p}{p}$$

The last row also assumed $34q^{35}$ is much less than 1. For these approximations to be valid, it suffices that p be larger than 0.13 (corresponding to about 133 ONEs in the register). The following table gives some values.

p	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2
E(n)	2.11	2.25	2.43	2.66	3.00	3.50	4.33	6.00

The meaning is evident — when you have plenty (more than 150) of ONEs expected, you surely encounter the first in the first few positions in the first row.

Neglecting the last approximation for formula (3) would leave

$$E(n) \cong \frac{1+p}{p} - 34q^{34} \qquad (4)$$

This approximation remains valid for q^{31} less than 1/10, corresponding to p larger than 0.07. For p = 0.07, formula (4) gives E(n) = 12.4.

The value p = 0.07 corresponds to an expectation of about 72 ONEs in the register. It has not been practical to work out a detailed approximation for smaller values of p, corresponding to fewer ONEs. Instead, consider the situation in which it is known that there is exactly one ONE in the register. If all locations are equally likely, the average number of steps needed is

$$E(n) = \frac{1}{1024} (2 + 3 + \ldots + 33 + 3 + 4 + \ldots + 34 + \ldots + 33$$
$$+ 34 + \ldots + 64$$
$$= 33$$

Assuming that the last estimate applies equally well to the situation in which the probability that an arbitrary bit is a ONE is 1/1024, there remains the question of values of E(n) for p between 0.001 and 0.07. It is evident that E(n) increases as p decreases, and that the variation should be fairly smooth. Accordingly, the data estimated thus far are plotted on a log-log chart shown as Figure B-1. A smooth curve has been sketched through them; it should suffice for estimation purposes. Note that the time required to resolve a response is found as a function of p as (E(n) x .016) μsec.

B.1.1 A Worst-Case Maximum Search

It is obvious that one aspect of the worst case in a maximum search is that the largest numerical value being sought appears in two distinct words of the memory. The next question is: which numerical value gives the largest search time ? The relationship between search time and the numerical value is most simply developed from an example.

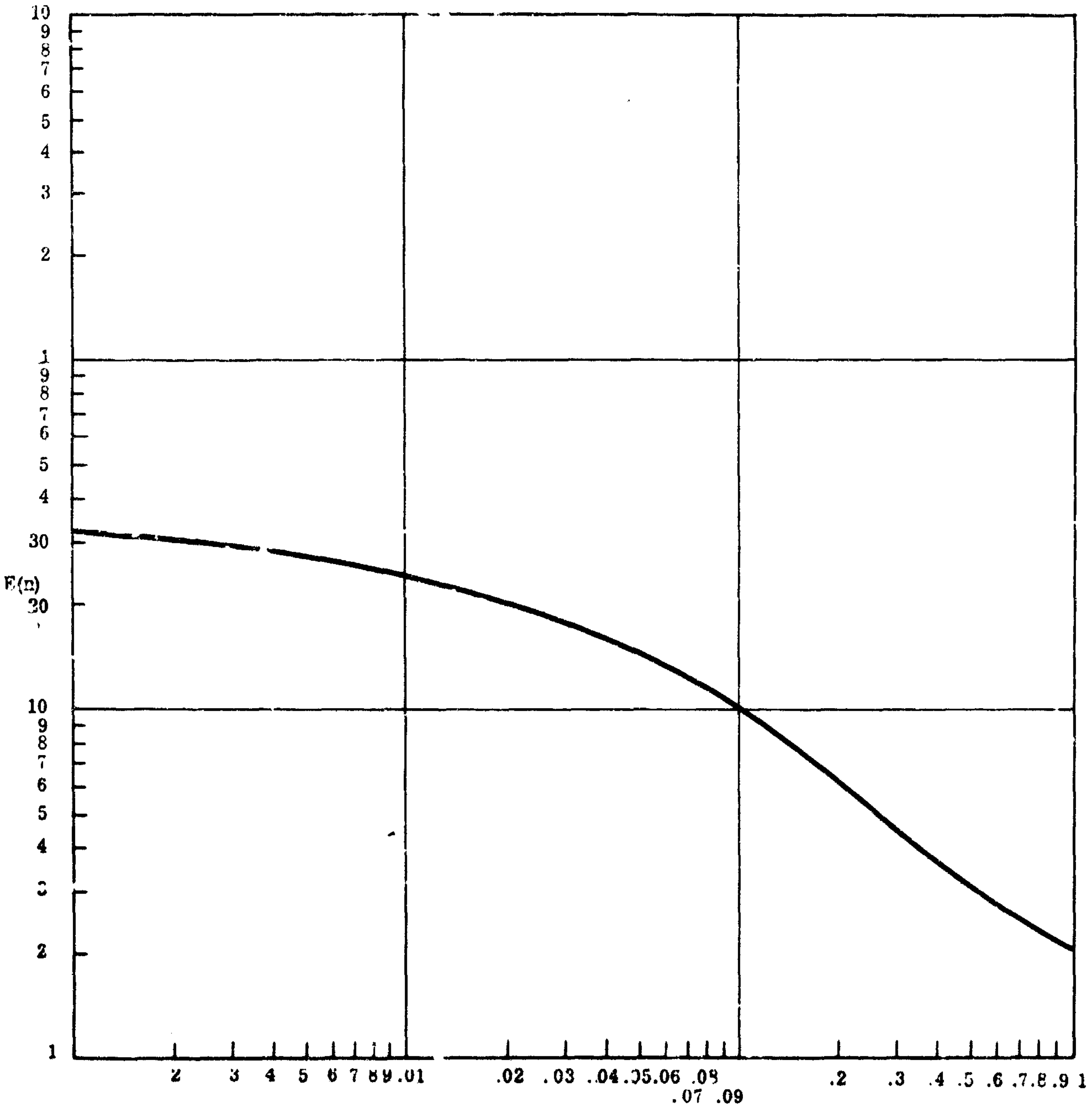


Figure B-1. Resolution Steps vs. Probability of ONE in Given Position of S

Let the maximum value, appearing at least twice in the memory, be 00101‘' (The pattern of the bits omitted is irrelevant.) The search procedure inspects bit 47 (highest order), and the resulting vector in the S-register is all ZERO's. Accordin' bit position 47 in the mask register is set to ZERO, indicating that for the remainde the maximum search, bit position 47 is to be skipped. The instruction now makes a second pass at memory, with a similar result for bit 46. On the third pass, howeve bit position 45 yields at least two ONE's. The instruction proceeds at once to inspec position 44. In this worst case, the procedure continues until bit position 0 is inspe for the first time. The nature of the termination of the instruction now depends on this last position yields a unique maximum (in this example, it doesn't) but the time involved for the termination is assumed insignificant. The time of interest is the um of time spent in inspections and in skipping positions.

The procedure is analyzed by using an illustration as shown below. The nu in the top row is the actual maximum of the numerical values in the mem . Each succeeding row shows the procedure for one pass at the memory. Each pass before t last ends as soon as a ZERO is encountered in the maximum. In our example, the fi pass inspects position 47, and finds a ZERO. The inspection is indicated by an "I" in the first row below the number. The second pass skips position 47 ("S" begins the se row), inspects position 46, and finds a ZERO. The third pass skips positions 47 and and finds a ONE

0	0	1	0	1	0	. . . 0	0	
I								I
S	I							I+S
S	S	I	I					2I+2S
S	S	I	S	I	I			3I+3S
-	.	-	-	-	.	- - -		.
S	S	I	S	I	S	- - - I		.
S	S	I	S	I	S	- - - S	I	(38-k)I+kS

in position 45. The pass does not end here; it goes on to possition 44, where a ZERO ends it.

The number of skips and inspections in each row is recorded at the right of the illustration above. In order to identify the worst case, we note the following:

(1) Every bit position is inspected at least once, the first time it is encountered.

(2) Bit position 0 is inspected just once. No matter what it contains, it has no other effect on instruction timing.

(3) The number of passes required is k+1, where k is the number of ZERO's in bit positions 47 through 1. The number of skips in these passes are 0, 1, 2, ..., k. Thus, the total number of skips in the instruction are $1+2+\ldots+k = \binom{k}{2}$.

(4) For each bit position containing a ONE, the number of re-inspections is equal to the number of passes after the first inspection, hence to the number of ZERO's in lower order positions (other than position 0). Thus, for a fixed value of k the largest number of re-inspections occurs when the 47-k ONE's occupy the 47-k bit positions of highest order. The total number of re-inspections is then k (47-k).

(5) Combining the foregoing remarks gives the result that for fixed k, the worst case involves 48 + k (47-k) inspections and $\binom{k}{2} = \frac{k(k-1)}{2}$ skips. If a skip takes 0.05 microsecond and an inspection takes 1.6 microsecond (including resolution time of 1.5 microsecond to detect the second responder), this worst case takes $-1.575k^2 + 74.975k + 77$ microseconds. This expression, in turn, is a maximum for k = 24; its value is about 980 microseconds.

B.1.2 A Worst-Case Search for Next-Less-Than

All of the analysis for the worst-case maximum search apply, with the following embellishments. Each pass is preceded by a less-than comparison of the entire memory with the test word. The result of the comparison is stored in the S-register, where it provides basis for eligibility. Each comparison requires approximately five microseconds. It has already been noted that k+1 passes are needed. Therefore, in terms of k, the worst case must involve

$$- 1.575k^2 + 74.975k + 77 + 5\,(k+1)\mu\text{sec.}$$
$$\cong (-1.575k^2 + 80k + 82)\ \mu\text{sec.}$$

The maximum would occur for k=25. For k=25 we get a maximum time of about 1100 microseconds.

B.1.3 Representative Times for Maximum Search and for Next-Less-Than

The average time spent in executing the maximum-search instruction or the next-less-than instruction depends strongly on the kind of data which may be expected in the memory. This fact is of primary importaace in any estimation of the running time of an actual program. In order to get some sort of representative value, it is assumed here that ZERO and ONE are equally likely in any bit position in the memory. A precise estimate of the instruction times, even in this case, has not been possible, but a rough estimate has been derived.

For a maximum search, the probability of finishing immediately upon inspecting bit position 48 is the probability that exactly one word starts with a ONE, or $1000/2^{1000} \cong 10^{-297}$. This is negligible.

The probability that a selected word starts with ten consecutive ONE's is $1/2^{10}$. The probability that <u>no</u> word starts with ten consecutive ONE's is $(1-1/2^{10})^{1000} \cong 1/e \cong 0.368$. The probability that exactly one word starts with ten consecutive ONE's is

$$\frac{1000}{2^{10}} \left(1 - \frac{1}{2^{10}}\right)^{999} \cong \frac{1}{e}$$

The probability that exactly two words start with ten consecutive ONE's is

$$\binom{1000}{2}\left(\frac{1}{2^{10}}\right)^{2} \left(1 - \frac{1}{2^{10}}\right)^{998} \cong \frac{1}{2e}$$

The probability that more than two words start with ten consecutive ONE's is approximately $1 - (1 + 1 + 1/2) / e \cong 0.08$. This probability will be neglected.

Now, if exactly one word starts with ten consecutive ONE's, the maximum search will terminate after somewhere between one and ten inspections. A good estimate of the instruction time in this case is 3.0 microseconds. If exactly two words start with ten consecutive ONE's, the search will end as soon as they differ. The probability that it happens in bit position 38 is 1/2. The probability that it happens in position 37 is 1/4, but in this case a ONE or a ZERO in position 38 is equally likely. The former case involves, in all, 6.6 microseconds, while the latter involves 12.65 microseconds. Still assuming that exactly two words started with ten consecutive ONE's, it is assumed without further analysis, that the average effects of words which still match through bit position 36 is negligible. For the cases already considered, the average instruction time is

$$(1/2)\ (6.0) + (1/8)\ (6.6) + (1/8)\ (12.65) \cong 5.43\ \mu\text{sec.}$$

The net contribution to the estimated running time of the cases where one or two words start with ten ONE's is

$$(1 \times 3.0 + 1/2 \times 5.43)/e \cong 5.77\ \mu\text{sec.}$$

The probability that one or two words start with ten ONE's is $(1 + 1/2)/e \cong 0.55$.

Now the probability that no word starts with either ten ONE's or with nine ONE's and one ZERO is, as before

$(1 - \frac{11}{2^{10}})^{1000} = e^{-11}$, which is very small. We conclude that in all significant cases, at least nine of the first ten bits of the maximum are ONE's. By analogy with the case of ten ONE's, it is assumed, without formal analysis, that a unique maximum is the predominant case when there are nine ONE's and one ZERO. The time for a maximum search in this case is between 0.2 microsecond and 10.85 microseconds. We use 6.0 microseconds as representative. Thus, to the previous cases we add $6 \times \frac{1}{e} \cong 2.82$ microseconds, getting a little over 8.5 microseconds as the overall representative time for a maximum search.

This time remains representative if the maximum search is masked provided that at least twelve bit positions are not masked.

For a search for next-less-than, the average time also depends on the criterion word. If this word is all ONE's, the estimate for a maximum search applies, except that if a ZERO is encountered in a bit position in the memory another five microseconds must be added. However, the probability of encountering a ZERO before finding the maximum is small.

If the criterion wo~~r~~d assumes an "average" value, it will eliminate about half of the contenders in memory. In the scope of the present effort, it has proved impossible to get a good estimate for the typical time for a next-less-than instruction in this case. Accordingly, engineering judgment has been invoked.

It is judged that a representative situation will involve four passes through the memory and require an average of about 29 microseconds. This estimate, again, is relatively insensitive to masking.

APPENDIX C. ROUTINES

In order to investigate various aspects of the AM system, such as comparative timing and programming ease, several routines were chosen to be coded in detail, both for GAP and for the 1604-B. Many of these routines were taken from the query subsystem and thus provided a mechanism with which to estimate timing of the representative queries.

The routines and the function of each are listed below:

(1) AND - To find the conjunction of two lists.

(2) DIST - To compute the nautical distance between two points given in latitude-longitude coordinates.

(3) DN - To eliminate double negatives in an expression written in Polish prefix form.

(4) EQU - To select related logical names from a data file directory.

(5) MAX(MIN) - To find the maximum (minimum) value of a descriptor.

(6) NGT(NLT) - To find the value of a descriptor next greater than (next less than) a given value.

(7) OR - To find the logical disjunction of two lists.

(8) REL - To select logical names of records which satisfy a requested attribute-descriptor relationship.

(9) RNR - To read next responder.

(10) SPO - To perform search operations on specified subsets of AM locations.

(11) TIME - To compute the time required to traverse a distance.

(12) VLLU - To perform a variable length lookup.

Where pertinent, AM coding and non-AM coding are provided for each routine along with a generalized timing formula. Comments are also provided with the coding.

In the timing formula of the routines in GAP, the average elapsed time assuming no I/O was used. For the 1604-B, the average execution time was used. In order to compute representative timing, typical values were assigned to the parameters found in the timing formulas.

The following assumptions were therefore made:

(1) There are 32 records in a data file.

(2) A data record contains on the average 32 words.

(3) The average number of responders per search is four.

(4) There are 200 records in a file directory.

(5) A file directory record contains on the average five words.

Preceding the coding of each routine is, where pertinent, a standard form which consists of the following information.

- SUBROUTINE NAME
- PURPOSE
- MAJOR PERFORMANCE LOCATION
- INPUT DESCRIPTION
- OUTPUT LIST
- COMMAND LIST
- TIMING FORMULA (GAP and 1604-B)
- REPRESENTATIVE TIMING (GAP and 1604-B)

SUBROUTINE NAME: AND

PURPOSE: To find the conjunction of two lists of logical names.

MAJOR PERFORMANCE LOCATION: Associative Memory

INPUT DESCRIPTION: Two lists of logical names

OUTPUT LIST: A list of logical names which are common to both input lists; these lists are not sorted.

COMMAND LIST: (See Table 7-1, Volume II, for a complete description of command list.

θ: The operation code is AND.

VV: L_1 contains the core address of where the starting address of a list of logical names is stored.

L_2 contains the core address of where the starting address of a second list of logical names is stored.

TIMING (GAP): Time (μsec) = $k \left[(49.4 + 7.1w_1) + (55.8 + 91.8w_2)\right]$

where:

k = no. of loads
w_1 = avg. no. of words in longer list/load
w_2 = avg. no. of words in shorter list/load

(Note: First expression in parentheses is AM load time; second expression is processing time excluding AM load time.)

TIMING (1604-B): Time (μsec) = $124.8 + w_2 (3.6w_1 + 18.4) + 13.2r$

where:

w_1 = avg. no. of words in longer list searched before finding a responder.

w_2 = no. of words in shorter list.

r = no. of responders entered into output list.

REPRESENTATIVE TIMING (msec):

GAP			1604-B
AM load time	processing time, excluding AM load time	Total	
1.4	0.9	2.3	1.7

SUBROUTINE NAME: AND

	GAP	
GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(1) Determine longer list, using size of list information stored in command list.	LDA	Load A-register with number of words in list.
	SUB	Subtract A from number of words in second list.
	AJP	Jump to instruction which inserts address of longer list into LDR.
(2) Transfer to AM operations.	SENSE SELECT FORCE	
(3) Load longer list into the AM.	EMY	Turn off busy bits.
	LDR	Set up S, R, N.
	RBL	Load words into AM.
(4) Load an element of shorter list into the comparand and match against longer list.	LDR EMC	
(5) If no responders, go to step (7).	JNR	
(6) Add address of responder to output list. Increment count of responders.	RDA ICI	
(7) Increment count of words in short list.	ICI	
(8) List completed.	JIH	Compare index with number of words in list.
(9) Store total count of responders in command list.	SIX	
(10) Halt AM.	HALT CLEAR	

SUBROUTINE NAME: AND

1604-B

GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(1) **Determine long and short list addressing.**	ENI	Set output list index.
	ENQ	Mask to isolate number of words.
	LDL	Isolate number of words.
	ARS	Position number of words.
	SAU	Save number of words.
	ENQ	Mask to isolate number of words.
	LDL	Isolate number of words.
	ARS	Position number of words.
	STA	Save number of words.
	ENA	Get number of words.
	SUB	Determine long and short list.
	AJP	Set long list address.
	ENI	Set short list address.
	ENI	Number of words in long list.
	ENI	Address of short list.
	ENA	Number of words in short list.
	INA	Set tally for short list.
	SAU	
	SLJ	
(2) **Search long list for responder.**	LDA	Logical name from short list.
	EQS	Compare against long list.
	SLJ	No responder.
(3) **Save responder address.**	ENA	Address of responder into output list.
	STA	
	INI	Increment output list index check for fini.
	ISK	
	SLJ	Number of words in output list.
	ENA	Position in A.
	ALS	Mask for number of words.
	ENQ	Total word → A.
	ADL	A → command list.
	STA	
	SLJ	

SUBROUTINE NAME: DIST

PURPOSE: To compute the nautical distance between two points given in latitude-longitude coordinates.

MAJOR PERFORMANCE LOCATION: Core

DATA INPUT: Dynamic ship file directory.

OUTPUT LIST: A pseudo data file which contains the following information for each record:

(1) Logical name — registry/ship serial number
(2) Attribute Identifier — DIST
(3) Descriptor — distance in nautical miles

COMMAND LIST: (See Table 7-1, Volume II, for complete description of command list)

θ: The operation code is DIST
VV: L_1 contains the core address of where the coordinates of a location are stored.

TIMING (1604-B): An approximation to the distance formula is 11 milliseconds.

SUBROUTINE NAME: DIST

1604-B

GENERAL PROCESSING PROCEDURE:

An approximation to the problem of determining the distance between two points, given their latitude and longitude, would be to consider the earth as a sphere. Such an approach is taken solely for the purpose of estimating 1604 time in computing distance. For more accurate distance calculations see Formulas and Tables for the Computation of Geodetic Positions, Special Publication No. 8, U.S. Department of Commerce, Coast and Geodetic Survey, Washington, D.C.

In the spherical system, the distance between two points P and Q, given their latitude and longitude, is:

$$\text{DIST (P,Q)} = \text{DCOS}^{-1} \left(R^2 - L^2/4\right)^{1/2} / \ R$$

where:

D = diameter of the earth
R = radius of the earth
L = length of the chord connecting P and Q

The chord length expressed in terms of latitude and longitude of the two points is:

$$L = R \left[(\text{COS } \phi_2 \ \ \text{COS } \theta_2 - \text{COS } \phi_1 \text{ COS } \theta_1)^2 + (\text{COS } \phi_2 \text{ SIN } \theta_2 - \text{COS } \phi_1 \text{ SIN } \theta_1)^2 + (\text{SIN } \phi_2 - \text{SIN } \phi_1)^2 \right]^{1/2}$$

where:

θ_1 = longitude of point P
ϕ_1 = latitude of point P
θ_2 = longitude of point Q
ϕ_2 = latitude of point Q
R = radius of the earth

SUBROUTINE NAME: DN

PURPOSE: To eliminate double negatives in an expression written in Polish prefix form.

MAJOR PERFORMANCE LOCATION: Associative memory

DATA INPUT:

(1) An expression in Polish prefix notation
(2) The symbol used for "not"

OUTPUT LIST: The same expression in Polish prefix notation with all double negatives eliminated.

CALLING SEQUENCE: *DN, 1, n, s

where;

1 = starting 1604-B location where expression in Polish prefix form is stored.
n = no. of words in expression
s = symbol used for negation

***Note: Two versions have been coded: DN2, which utilizes busy bits; DN1, which does not utilize busy bits.**

SUBROUTINE NAME: DN1

GAP

GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(1) Load symbols sequentially into AM.	EMY	Turn off busy bits
	LDR	Set up S, R, N
	RBL	Load expression into AM
(2) Set response store for all "nots" in the expression.	LDR	Set up S, R, N
	EMC	
(3) If no responders, exit.	JNR	
(4) Advance response store one location. Perform exact match and AND results.	LDR	Set up S, R, N
	EMC	Set T = 1, Z = 1
(5) If no responders, exit.	JNR	
(6) Read contents of first responder.	LDR	Set up S, R, N
	RCF	
(7) The λ portion of the responder is used to set up a transfer that will transfer a 1604 taken substring from λ + 1 into AM location λ -1 through the end of the string.*	LDI	Set up B1
	ICI	Increment B1 by 1
	LDI	B2 received relative of responders.
	DEI	Step B2 down 1
	SIX	Store index B2 in temp. location for "R" LDR
	LDR	Set up for WFA
	WFA	Write 1604-B location of LDR instruction to be modified into first available AM address
	LDR	Set up for EMC
	EMC	Do exact match on LDR
	LDR	Set up for WIR
	WIR	Write AM address "R" through mask into responder.
	LDR	Set up for RCF
	RCF	Read contents of first responder into 1604-B location of LDR instruction to be modified.
	LDR	LDR to be modified, sets up "R", S, N, for RBL
	RBL	Load list of tokens for 1604
	JIH	If B-Box not MAX + 1, loop back to step (2); otherwise, exit.
(8) Halt AM	CLEAR	

* Refers to the address of the element being scanned.

SUBROUTINE NAME: DN2

GAP

GENERAL PROCESSING PROCEDURE		INSTRUCTION	COMMENTS
(1)	Load AM with token list, 1.	EMY	Turn off busy bits.
		LDR	Set up S, R, N.
		RBL	
(2)	Set RS for all "nots".	LDR	Set up 3, R, N.
		EMC	Do exact match on symbol S.
(3)	Advance response store. Perform exact match and AND results.	LDR	Set up S, R, N.
		EMC	Set T = 1, Z = 1.
(4)	If no responders, exit.	JNR	
(5)	Store count of responders.	LDR	
		RCR	
(6)	Store address of responders in 1604-B.	LDR	Set up S, R, N.
		RDA	
(7)	Modify R of LDR with address of responders, and reset busy bits of responder and responder + 1.	LDI	Load B3 with address of a responder.
		ICI	Increment B2 index tag for indicating address of responder.
		DEI	Decrement B3, address of 1st "not" in a "DN".
		SIX	Store AM address in B3 in a 1604-B location (use B2 as the index tag for storing).
		LDR	Set up WFA.
		WFA	Write 1604-B location of LDR instruction to be modified into first available AM address.
		LDR	Set up EMC.
		EMC	Do exact match on LDR instruction to be modified.
		EMC	Set up WIR.
		WIR	Write AM address through mask into responders.
		LDR	Set up RCF.
		RCF	Read contents of first responder into 1604-B location of LDR instruction to be modified.
		LDR	LDR with R modified and N = 2.
		RBF	Change busy bit status of "DN".
		ICI	Increment counter of number of DN's.
		JIH	If all DN's have not been processed, return to beginning of step (7).

SUBROUTINE NAME: DN2

GAP (Contd')

GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(8) Store all active tokens in 1604-B.	LDR	Set up S, R, N.
	MMC	Store all tokens in responders (DN's will be ignored since their busy bit status is 0).
	RDC	Store desired responders.
(9) Halt AM.	CLEAR	

SUBROUTINE NAME: EQU

PURPOSE: To select related logical names of particular logical names from a data file directory.

MAJOR PERFORMANCE LOCATION: Associative memory

INPUT DESCRIPTION: Data file directory as described in 1231-TN-11.

OUTPUT LIST: Logical names of related records

COMMAND LIST: (See Table 7-1, Volume II, for a complete description of command list.)

θ: The operation code is EQU.
VV: L_1 contains the core address of where the starting location of a list of logical names is stored.

TIMING (GAP): Time (μsec) = $k \left[(49.4 + 7.1w) + (14.0 + e\,(93.1 + 3.6nr))\right]$

where:

w = avg. no. of words located into AM/load
e = avg. no. of logical names in L_1.
nr = avg. no. of records searched before finding a match on logical names per load.

(Note: First expression in parentheses is AM load time; second expression is processing time excluding AM load time.)

TIMING (1604-B): Time (μsec) = $44.2 + e\,(70.0 + 3.6w) + 51.4x$

where:

e = avg. no. of logical names in L_1
w = avg. no of words searched before finding a match on logical name.
x = no. of related logical names put into output list.

REPRESENTATIVE TIMING (msec):

GAP			1604-B
AM load time	processing time, excluding AM load time	Total	
7.3	3.6	10.9	7.1

SUBROUTINE NAME: EQU

	GENERAL PROCESSING PROCEDURE	GAP INSTRUCTION	COMMENTS
(1)	Transfer to AM operations.	RESUME	
(2)	Load the particular data file directory into AM.	EMY LDR RBL	Turn off busy bits Set up S, R, N Load words into AM
(3)	Perform an exact match on logical names.	LDR EMC	h contains starting location of L_1; q_2 indexes sequential logical names.
(4)	Store address of responder.	RDA	
(5)	Increment q_2.	ICI	
(6)	List completed.	JIH	Compare index with number of words in list. If completed, go to step (7) otherwise, go to step (3).
(7)	Halt AM.	CLEAR	
(8)	Obtain address of logical name of related records through a search on logical name address table.	THS	
(9)	Store logical names and count in output list.	LDA STA	

SUBROUTINE NAME: EQU

1604-B

GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(1) Initial Housekeeping.	ENA	L1.
	INA	Number of words in logical name
	SAU	list.
	ENI	Set tally.
	ENA	Initialize responders index.
	SAU	Initialize to L1.
	ENI	Initialize address of logical
	ENI	name list.
		Number of words in file directo.y.
(2) L1 equal to file directory logical name.	LDA	Logical name.
	EQS	Equal to L1.
	SLJ	No.
	RAO	Increment to related logical name.
(3) Update output list.	LDA	Move related logical name to
	STA	output list.
	RAO	Increment to next logical name.
	INI	Increment responders index.
(4) Determine end of related logical names.	ENQ	Logical name mask → Q.
	ENA	Check for logical name.
	MEQ	
	SLJ	
	SLJ	
(5) Search for next logical name.	ENA	Beginning of file directory.
	SAU	End of file directory.
	ENA	Determine length of file
	SUB	directory.
	ENI	Logical name flag.
	ENA	Logical name mask.
	ENQ	Search for next logical name.
	MEQ	
	SLJ	
	SLJ	
(6) Check for end of L1 list.	ISK	Tally on L1 list.
	SLJ	

1604-B (Contd.)

GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(7) Number of respoaders into the command list.	ENA	Number of responders in output
	ALS	list.
	ENQ	Position A
	ADL	Mask for A
	STA	Command list plus number of
	SLJ	responders.
		A into command list.
		Exit.

SUBROUTINE NAME: MAX(MIN)

PURPOSE: To find the maximum (minimum) value of a descriptor.

MAJOR PERFORMANCE LOCATION: Associative memory.

INPUT DESCRIPTION: Data records as described in 1231-TN-11.

OUTPUT LIST: Logical name of the record having the maximum (minimum) attribute value.

COMMAND LIST: (See Table 7-1, Volume II, for a complete description of command list)

● The operation code is MAX(MIN) n, where n indicates the number of words in the descriptor.

V: L_1 contains the core address of where the attribute identifier is stored.

TIMING (GAP): Time (μsec) = $k \left[(49.4 + 7.1w) + (330 + 14.2r + 3.6nr)\right]$

where:

k = no. of loads
w = avg. no. of words loaded into AM/load
r = avg. no. of responders/load
nr = avg. no. of data records/load

(Note: First expression in parentheses is AM load time; second expression is processing time, excluding AM load time.)

TIMING (1604-B): Time (μsec.) = $\left[214.4 + 3.6w + nr\,(103.0 + 3.6w) + 13.2r\right]$

where:

w = avg. no. of words in record searched before finding match on attribute
nr = no. of data records
r = no. of potential responders

REPRESENTATIVE TIMING (msec.):

GAP			1604-B
AM load time	processing time, excluding AM load time	Total	
7.3	.5	7.8	5.7

SUBROUTINE NAME: MAX(MIN)

	GENERAL PROCESSING PROCEDURE	GAP INSTRUCTION	COMMENTS
(1)	Transfer to AM operations.	RESUME	
(2)	Load the data records into the AM.	EMY LDR RBL	Turn off busy bits. Set up S, R, N. Load words into AM.
(3)	Load L_1 into the comparand and perform an exact match.	LDR EMC	
(4)	If no responders. exit.	JNR	
(5)	Advance the response store one position and store the address of responders in core.	RDA	T=1
(6)	Count number of responders.	RCR	Use count to determine number of words to be loaded in step (7).
(7)	Load the list of selected attribute values into AM.	EMY LDR RBL	Turn off busy bits. Set up S, R, N. Load only descriptors.
(8)	Find max (min).	MAX(MIN)	
(9)	Count number of responders.	RCR	
(10)	Store count of responders in index register.	SIX	
(11)	Test for count greater than 1.	JIH	If count greater than 1, then test for n greater than 1; otherwise, go to step (16).
(12)	Test for all n processed.	ICI JIH	If n 1, go to step (13); otherwise, go to step (16).
(13)	Advance the response store one position and store the address of responders in core.	RDA	T=1
(14)	Load next word of n-word descriptor response into AM.	EMY LDR RBL	Turn off busy bits. Set up S, R, N. Load only descriptors.
(15)	Jump to step (8).	JUC	

SUBROUTINE NAME: MAX(MIN)
(cont'd)

GENERAL PROCESSING PROCEDURE	GAP INSTRUCTION	COMMENTS
(16) Store address of responder(s).	RDA	
(17) Halt AM.	CLEAR	
(18) Perform search on logical name address table for responder.	THS	
(19) Store logical names in output list.	LDA STA	

SUBROUTINE NAME: MAX(MIN)

1604-B

GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(1) Initial housekeeping.	ENI	Responders index.
	ENA	Beginning address of LN list.
	INA	Number of words ir LN list into tally
	SAU	First logical home address into output list.
	LDA	
	STA	Address of first LN address
	ENI	n Mask → Q.
	ENQ	L (Q)n → A.
	LDL	Save n.
	SAU	Set comparison address.
	ENA	Operator code mask → Q.
	SAU	L (Q)σ → A.
	ENQ	MAX or MIN.
	LDL	Set MAX or MIN addressing.
	AJP	
	ENA	
	SAU	
	ENA	
	SAU	
	ENA	
	SAU	
	SLJ	
(2) Find length of first logical name record.	LDA	Small LN address.
	INI	Index to next address.
	SUB	Greater address.
	INA	Inclusive numbers
	SAU	Save number of words.
	ENI	
(3) Get address of attribute and descriptors in first record.	LDA	Attribute from command list → A.
	EQS	Search record for attribute.
	SLJ	No attribute match.
	INI	Increase to descriptor.
	ENA	Save address.
	SAU	
	SLJ	
(4) No attribute match.	ISK	Check for end of logical name address list.
	SLJ	
	ENI	Clear number oi responders to Ø.
	SLJ	

1604-B

GENERAL PROCESSING PROCEDURE		INSTRUCTION	COMMENT
(5)	Find length of logical name record.		
(6)	Get address of attribute and descriptor in record.		
(7)	Switch for MAX or MIN.	SLJ	
(8)	Determine descriptor relationships.	LDA	Descriptor of current responder.
		EQS	Descriptor from record.
		SLJ	Not equal.
		IJP	All words of descriptor equal.
		LDA	Yes, add address to list.
		STA	Increase output index.
		INI	
		SLJ	
		INI	Increase descriptor addressing.
		RAO	
		SLJ	
		LDA	Descriptor of current responders.
		THS	Descriptor from record.
		SLJ	Set for MAX or MIN.
		SLJ	
		ENI	Clear output list index.
		ENA	Logical name address to output list.
		STA	
		ISK	Check for end of logical name list.
		SLJ	
(9)	Number of words in output list → command list.	ENA	Number of words in output list.
		ALS	Position in A.
		ENQ	Mask for number of words.
		ADL	Total words → A.
		STA	A → command list.
		SLJ	Exit.

SUBROUTINE NAME: NGT(NLT)

PURPOSE: To find the value of a descriptor next greater than (next less than) a given value.

MAJOR PERFORMANCE LOCATION: Associative memory.

DATA INPUT: Data records as described in 1231-TN-11

OUTPUT LIST: Logical name of the record having the next greater than (next less than) attribute value.

COMMAND LIST: (See Table 7-1, Volume II, for complete description of command list.)

φ: The operation code is NGT(NLT), n where n indicates the number of words in the descriptor.

VV: L_1 contains the core address of where the attribute identifier is stored.

L_2 contains the core address of where the starting address of the value to be compared is stored.

TIMING: (GAP): Time (μsec) = k (49.4 + 7.1w) + (357.3 + 14.2r + 3.6nr)

where:

k = no. of loads
w = avg. no. of words loaded into AM/load
r = avg. no. of responders/load
nr = avg. no. of data records/load

(Note: First expression in parentheses is AM load time; second expression is processing time, excluding AM load time.)

TIMING (1604-B): Time (μsec) = 276.2 + nr (168.6 + 3.6w) + 37.6r

w = avg. no. of words in record searched before finding match on attribute
nr = no. of data records
r = no. of potential responders

REPRESENTATIVE TIMING (msec):	GAP			1604-B
	AM load time	processing time, excluding AM load time	Total	
	7.3	.5	7.8	8.1

SUBROUTINE NAME: NGT(NLT)

GAP

GENERAL PROCESSING PROCEDURE		INSTRUCTION	COMMENTS
(1)	Transfer to AM operations.	RESUME	
(2)	Load the data records into the AM.	EMY LDR RBL	Turn off busy bits. Set up S, R, N. Load words into AM.
(3)	Load L_1 into the comparand and perform an exact match.	LDR EMC	
(4)	If no responders, exit.	JNR	
(5)	Advance the response store one position and store the address of responders in core.	RDA	T = 1
(6)	Count number of responders.	RCR	Use count to determine number of words to be loaded in step (7).
(7)	Load the list of selected attribute values into AM.	EMY LDR RBL	Turn off busy bits. Set up S, R, N. Load only descriptors.
(8)	Find the next greater than (next less than) valued descriptor.	LDR NHC(NLC)	h contains L_2.
(9)	Count number of responders.	RCR	
(10)	Store count of responders in index register.	SIX	
(11)	Test for count greater than 1.	JIH	If count greater than 1, then test for n greater than 1; otherwise, go to step (16).
(12)	Test for all n processed.	ICI JIH	If $n > 1$, go to step (13); otherwise, go to step (16).
(13)	Advance the response store one position and store the address of responders in core.	RDA	T = 1

SUBROUTINE NAME: NGT(NLT)
(cont'd)

	GENERAL PROCESSING PROCEDURE	GAP INSTRUCTION	COMMENTS
(14)	Load next word of n-word descriptor response into AM.	EMY	Turn off busy bits.
		LDR	Set up S, R, N.
		RBL	Load only descriptors.
(15)	Jump to step (8).	JUC	
(16)	Store address of responder(s).	RDA	
(17)	Halt AM.	CLEAR	
(18)	Perform search on logical name address table for responder.	THS	
(19)	Store logical names in output list.	LDA	
		STA	

SUBROUTINE NAME: NGT(NLT)

1604-B

GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(1) Initial housekeeping.	ENI	Responders index.
	ENA	Logical home address table.
	INA	Number of words in LN table into tally.
	SAU	
	ENI	Address of first LN address.
	ENQ	n Mask → Q.
	LDL	L (Q)n → A.
	SAU	Save n.
	ENA	Set comparison.
	SAU	Address.
	ENQ	Operation mask → Q.
	LDL	L (Q) opr → A.
	AJP	NGT or NLT.
	ENA	Set addressing for NGT or NLT.
	SAU	
	ENA	
	SAU	
	ENA	
	SAU	
	SLJ	
(2) Find length of first logical name record.	LDA	Small LN address.
	INI	Index to next LN address.
	SUB	Greater address.
	INA	Inclusive numbers.
	SAU	Save number of words in record.
	ENI	
(3) Get address of attribute and descriptor in first record.	LDA	Attribute from command list → A.
	EQS	Search record for attribute.
	SLJ	No attribute match.
	INI	Increase to descriptor.
	ENA	Save address.
	SAU	
	SLJ	
(4) No attribute match.	ISK	Check for end of logical name address list.
	SLJ	
	SLJ	
(5) Descriptors have desired relationship to input.	LDA	Descriptors equal.
	EQS	
	SLJ	
	SLJ	
	LDA	
	THS	
	SLJ	
	SLJ	

SUBROUTINE NAME: NGT(NLT)
(Cont'd)

		1604-B	
GENERAL PROCESSING PROCEDURE		INSTRUCTION	COMMENTS
(6)	Update output list.	ENA	Same address as descriptor.
		STA	Logical name address to output list.
		LDA	Increment responders index.
		STA	
		INI	
		ISK	Check for end of logical name lists.
		SLJ	
(7)	Number of words in output list → command list.	ENA	Number of words in output list.
		ALS	Position in A.
		ENQ	Mask for number of words.
		ADL	Total words → A.
		STA	A → command list.
		SLJ	
(8)	Find length of logical name record.		
(9)	Get address of attribute and descriptor.		
(10)	Descriptors have desired relationship to input.		
(11)	Descriptors have desired relationship to current output.	ENI	Clear responders index.
		SLJ	

SUBROUTINE NAME: OR

PURPOSE: To find the logical disjunction of two lists of logical names.

MAJOR PERFORMANCE LOCATION: Associative memory

INPUT DESCRIPTION: Two lists of logical names.

OUTPUT LIST: The union of both input lists.

COMMAND LIST: (See Table 7-1, Volume II, for complete description of command list.)

θ: The operation is OR

VV: L_1 contains the core address of where the starting address of a list of logical names is stored.

L_2 contains the core address of where the starting address of a second list of logical names is stored.

TIMING (GAP): Time (μsec) = $k \; [(49.4) + 7.1 \, w_1) + (75.8 + 93.1 \, w_2)]$

where:

k = no. of loads
$w1$ = avg. no. of words in longer list/load
w_2 = avg. no. of words in shorter list/load

(Note: First expression in parentheses is AM load time; second expression is processing time,excluding AM load time.)

TIMING (1604-B): Time (μsec) = $132.0 + w_2 \, (3.6 w_1 + 24.0) + 24.6 r$

where:

w_1 = avg. no. of words in longer list searched before finding a responder.
w_2 = no. of words in shorter list.
r = no. of times the longer list is searched finding no responders.

REPRESENTATIVE TIMING (msec.):

GAP			1604-B
AM load time	processing time, excluding AM load time	Total	
1.4	.9	2.3	1.7

SUBROUTINE NAME: OR

		GAP	
	GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(1)	Determine longer list, using size of list information stored in command list.	LDA SUB AJP	Load A-register with number of words in list. Subtract A from number words in second list. Jump to instruction which inserts address of longer list into LDR.
(2)	Transfer to AM operations.	RESUME	
(3)	Load longer list into the AM.	EMY LDR RBL	Turn off busy bits. Set up S, R, N. Load words into AM.
(4)	Load index register with size of longer list.	LDI	
(5)	Load an element of shorter list into the comparand and match against longer list.	LDR EMC	
(6)	If no responders, add element to longer list and add one to counter.	JNR RDA ICI ICI	Counter for number of elements in shorter list. Counter for union of lists.
(7)	List completed.	JIH	Compare index with number of words in list.
(8)	Store total count of responders in command list.	SIX	
(9)	Halt AM.	CLEAR	

SUBROUTINE NAME: OR

1604-B

GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(1) Determine long and short list addressing.	ENQ	Mask to isolate number of words.
	LDL	Isolate number of words.
	ARS	Position number of words.
	SAU	Save number of words.
	ENQ	Mask to isolate number of words.
	LDL	Isolate number of words.
	ARS	Position number of words.
	STA	Save number of words.
	ENA	Get number of words.
	SUB	Determine long and short list.
	AJP	Set short list address.
	ENI	Number of words in long list.
	ENI	Set long list address.
	ENI	Save beginning address.
	SIU	Get next output address.
	INI	Short list address.
	ENA	Number of words in short list.
	INA	Set tally for short list.
	SAU	
	SLJ	
(2) Search long list for responder.	LDA	Logical name from short list.
	EQS	Compare against long list.
	SLJ	No responder.
(3) Check for end-of-short list.	ISK	Tally on short list.
	SLJ	Loop on short list.
(4) Move number of words in long (output) list→ command list.	ENA	Number of words in long list.
	ALS	Position in A.
	ENQ	Mask for number of words.
	ADL	Total words → A.
	STA	A → command list.
	SLJ	Exit.
(5) Move element with no responder → long (output) list.	LDA	Logical name from short list
	STA	into long list.
	INI	Increase long list index.
	SLJ	Go tally on short list.

SUBROUTINE NAME:	REL
PURPOSE:	To select logical names of records which satisfy a requested attribute-descriptor relationship.
MAJOR PERFORMANCE LOCATION:	Associative Memory
DATA INPUT:	Data records as described in Paragraph 4.4, Volume II.
OUTPUT LIST:	Logical names of records satisfying selection criteria*
COMMAND LIST:	(See Table 7-1, Volume II, for complete description of command list.)
●:	The operation code is RELi, n where i indicates the particular relationship and n indicates the number of words in the descriptor. The relationship (this list is by no means exhaustive and can be expanded) and their respective codes are: 1 = equality 2 = inequality 3 = less than 4 = greater than 5 = less than or equal 6 = greater than or equal to
VV:	L_1 contains the core address of where an attribute identifier is stored.* L_2 contains the core address of where the value of the attribute, i.e., the descriptor, is stored.

TIMING (GAP):	Time (μsec) = 50 + k [(13.6 nr) r + 25.5r + 7.1w + 52.4n + 135)]
	k = no. of loads
	nr = avg. no. of data records/load
	r = avg. no. of responders/load
	w = avg. no. of words loaded into AM/load
	n = no. of words in descriptor

TIMING (1604-B):	Time (μsec) = 86.2 + nr (3.6w + 120.6) + 17.4r where: w = avg. no. of words searched before finding match on attribute. nr = no. of data records. r = no. of responders.

*Note: If L_1 contains logical names, then the entire data record will be selected.

REPRESENTATIVE TIMING (msec.):	AM load time	GAP processing time, excluding AM load time.	Total	1604-B
	7.3	2.0	9.3	5.9

SUBROUTINE NAME: REL

GENERAL PROCESSING PROCEDURE		GAP INSTRUCTION	COMMENTS
(1)	Modify the AM instruction to perform the requested relationship.	LDA	Load A register with opcode, RELi.
		LDQ	Load Q register with mask.
		MEQ	Determine i.
		ENA	Store i.
		AJP	Jump to instruction which inserts requested AM instruction.
(2)	Transfer to AM operations.	RESUME	
(3)	Load the data records into the AM.	EMY	Turn off busy bits.
		LDR	Set up S, R, N.
		RBL	Load words into AM.
(4)	Do an equality search on attribute identifier.	LDR	
		EMC	
(5)	If no responders, exit.	JNR	
(6)	Advance response store. Perform particular description relationship and AND results.	for i=1 EMC	Set T = 1, Z = 1.
		= 2 MMC	
		= 3 LTC	
		= 4 GTC	
		= 5 LEC	
		= 6 GEC	
(7)	Perform step (4), n-1 times.	ICI	Increment index and compare with n, number of descriptor words.
		JIH	
(8)	If no responders, exit.	JNR	
(9)	Store address of responders.	RDA	
(10)	Halt AM.	CLEAR	
(11)	Perform search on logical name address table for each responder.	THS	
(12)	Store logical names in output list.	LDA	
		STA	

GENERAL PROCESSING PROCEDURE	1604-B INSTRUCTION	COMMENTS
(1) Initial housekeeping.	ENI	Set logical name list counter.
	ENQ	i Mask → O.
	LDL	L (Q) i → A.
	SAU	Save i.
	ENQ	n mask → Q.
	LDL	L (Q) n → A.
	SAU	Save n.
	ENI	Set logical name address table
	ENI	index.
	ENA	Initialize output table index.
	SAU	Set comparison address.
(2) Find length of logical name record.	LDA	Small LN address.
	INI	Increment to next address.
	LDA	Smaller address → A.
	SUB	Greater address.
	INA	Inclusive numbers.
	SAU	Save number of words in record.
	ENI	
(3) Get address of attribute and descriptor in record.	LDA	Attribute from command list → A.
	EQS	Search record for attribute.
	SLJ	No attribute match.
	INI	Increase to descriptor.
	ENA	Address saved.
	SAU	Number of words in descriptor.
	ENI	
(4) Determine command descriptor relationship.	LDA	(L2) → A.
	EQS	(L3) = Descriptor.
	SLJ	No.
	IJP	Yes - all of descriptor =
	ENA	Yes - i → A.
	INA	A - 1 → A.
	AJP	A = ø select record
	INA	A - 4 → A.
	AJP	A = + select record.
	SLJ	Not preferred.
	INI	Increase descriptor address.
	RAO	Increase descriptor address.
	SLJ	Loop on descriptor.
	ENA	i → A.

SUBROUTINE NAME: REL
(cont'd)

1604-B

GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
	LRS	$A_i \rightarrow Q_{47}$.
	LDA	(L2) → A.
	THS	(L2) < descriptor.
	SLJ	Check i.
	QJP	Q = (-) preferred.
	SLJ	Not preferred.
	QJP	Q = (-) not preferred.
(5) Store logical name of selected record into output list.	LDA	Logical name.
	STA	To output area.
	INI	Incr ease output index.
(6) End of loop housekeeping.	ISK	Check for end of logical name.
	SLJ	Address list — loop.
(7) Number of words in output list → command list.	ENA	Number of words in output list.
	ALS	Position in A.
	ENQ	Mask for number of words.
	ADL	Total words → A.
	STA	A → command list.
	SLJ	Exit.

MACRO NAME:	RNR — READ NEXT RESPONDER
PURPOSE:	To read out data associated with a match key where variable formats are used; e.g., to read out the logical name of a record chosen by a match on another field.
MAJOR PERFORMANCE LOCATION:	Associative memory.
DATA INPUT:	Data records as described in Paragraph 4.4, Volume II
OUTPUT LIST:	Result of RNR operation
MACRO ARGUMENT:	N - B = before AM location - A = after AM location A - XXXX AM location R - 1 = restore - Ø = do not restore

TIMING (μsec) - GAP:	Before: 318.1 + 7a + 7r
	After: 197.8 + 4b
	a = no. of "after" responders
	b = no. of "before" responders
	r = no. of responders

REPRESENTATIVE TIMING (msec):	Before: 0.4 After: 0.2

SUBROUTINE NAME: RNR

GAP

GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(1) Isolate restore flag.	ENQ	Restore mask.
	LDL	Isolate restore flag.
	STA	Save flag.
(2) Determine option.	ENQ	Option mask.
	LDL	Isolate option.
	SUB	
	AJP	
(3) Initialize for "after" option.	SENSE	Transfer to AM operations.
	SELECT	
	RESUME	
(4) Inhibit all "before" responders.	LDR	Set R and N.
	RBE	Eliminate "before" responders
(5) Read contents of first responder.	LDR	Set S.
	RCF	
(6) Determine restore option.	LDI	Get flag.
	JIH	Determine option.
(7) Restore.	LDR	Set R, N.
	ACT	Activate.
	JUC	Jump to exit housekeeping.
(8) Initialize for "before" option.	LDA	Determine N for future RBE and activate instructions.
	SUB	
	INA	
	STA	
(9) Transfer to AM operations.	SENSE	
	SELECT	
	RESUME	
(10) Eliminate "after" responders.	LDR	Set R and N.
	RBE	Random block erase.

SUBROUTINE NAME: RNR
(cont'd)

GENERAL PROCESSING PROCEDURES		GAP INSTRUCTION	COMMENTS
(11)	Erase all but last responder.	RCR	Count of responders equals C.
		LDI	∅ into index.
		ICI	Add 1 to index.
		EFR	Erase first responder.
		JIH	Jump if index equals C.
		JUC	Loop to increment index.
(12)	Read contents of the remaining responder.	LDR	
		RCF	
(13)	Determine restore option.	LDI	Get flag.
		JIH	Determine option.
(14)	Load buffer into response store.	LDR	Use mask of zeroes.
		WCR	
(15)	Activate.	LDR	
		ACT	
(16)	Halt AM operations.	CLEAR	Exit.
		SLJ	

MACRO NAME: SPO - SELECTED PERFORM OPERATIONS

PURPOSE: To perform search operations on specified subsets of AM locations.

MAJOR PERFORMANCE LOCATION: Associative memory.

DATA INPUT: Data records as described in Paragraph 4.4, Volume II

OUTPUT LIST: Results of search operation

MACRO ARGUMENTS:
M = XXX operation to be performed
N = 1 - search responders
2 - search between addresses
θ_1 = beginning address (AM)
θ_2 = ending address (AM)

TIMING (μsec) - GAP:

Search Responders: $288.9 + 15.2r$
Search Between Addresses: $392.8 + 4NW + 7.1r$

NW = no. of words not searched
r = no. of responders

Search Responders: 0.4 msec.

Search Between Addresses: 3.0 msec.

SUBROUTINE NAME: SPO

		GAP	
GENERAL PROCESSING PROCEDURE		INSTRUCTION	COMMENTS
(1)	Determine operation to be performed.	ENQ	Operation mask.
		LDL	Isolate operation.
		ALS	Position operation.
		SAU	Save operation.
(2)	Determine search environment.	ENQ	Search mask.
		LDL	Isolate search designator.
		ARS	Position search designator.
		INA	Subtract 1.
		SAU	Save results.
		AJP	
(3)	Set up search operation code.	ENA	Get operation code.
		ENQ	Get operation mask.
		ADL	Set search instruction.
		STA	
		SLJ	
(4)	Branch on search environment.	ENA	
		AJP	
(5)	Initialize to search responders.	SENSE	Transfer to AM operations.
		SELECT	
		RESUME	
(6)	Yield all active nonresponders.	CMB	Complement buffers.
(7)	Set to Ø, busy bits of nonresponders.	EAR	Erase all responders.
			Advance buffer one position before operation (T = 1).
(8)	-Perform search specified, saving buffer.	LDR	Set S, R, N.
		XXX	Operation code set.
			Save buffer (Z = 3).
(9)	Get results of search and hold.	JNR	Jump on no responders.
		RCR	Read count of responders.
		RDA	Read addresses of responders.
(10)	Restore conditions.	CMB	Complement buffer.
		LDR	Set S, R, N.
		WCR	Write Ø into responders.
		JUC	Jump to exit housekeeping.

SUBROUTINE NAME: SPO
(cont'd)

	GENERAL PROCESSING PROCEDURE	GAP INSTRUCTION	COMMENTS
(11)	Initialize to search between addresses.	LDA SUB INA STA LDA SUB INA STA	Compute number of words in AM block preceding search area. Compute number of words in AM block following search area.
(12)	Transfer to AM operations.	SENSE SELECT RESUME	
(13)	Set busy bits to ø for AM block preceding search area.	LDR RBE	Set R and N. Random block erase.
(14)	Set busy bits to ø for AM block following search area.	RCR LDR RBE	Place word count in N field and set R and N. Random block erase.
(15)	Perform search specified.	LDR XXX	Set S, R, N. Operation code set.
(16)	Get results of search and hold.	JNR RCR RDA	Jump on no responders. Read count of responders. Read addresses of responders.
	Set busy bits to 1 for AM block preceding search area.	RCR LDR ATV	Place word count in N field and set R and N. Activate.
(18)	Set busy bits to 1 for AM block following search area.	RCR LDR ATV	Place word count in N field and set. R + N Activate.
(19)	Halt AM.	CLEAR SLJ	Exit.

SUBROUTINE NAME: TIME

PURPOSE: To compute the time required to traverse a distance.

MAJOR PERFORMANCE LOCATION: Core

DATA INPUT: Dynamic ship file as described in 1231-TN-11

OUTPUT LIST: A pseudo data file which contains the following information for each record:

(1) Logical name — latitude-longitude
(2) Attribute Identifier — TIME
(3) Descriptor — time to tenths of an hour

COMMAND LIST: (See Table 7-1, Volume II, for complete description of command list)

θ The operation code is TIME.

V: L_1 contains the core address of where the coordinates of a location are stored.

TIMING (GAP): $k \left[(49.4 + 7.1w) + 12000s \right]$

where:

k = no. of loads
s = avg. no. of ships/load

(Note: First expression in parentheses is AM load time; second expression is processing time, excluding AM load time.)

TIMING (1604-B): Time (μsec) = $56.8 + 3.6w_1 + s\,(3.6w_2 + 12200.)$

where:

w_1 = no. of words in file.
w_2 = avg. no. of words in records.
s = no. of ships in file.

REPRESENTATIVE TIMING (msec):	GAP			1604-B
	AM load time	processing time, excluding AM load time	Total	
	7.3	12.0	19.3	15.0

SUBROUTINE NAME: TIME

		GAP	
	GENERAL PROCESSING PROCEDURE	INSTRUCTION	COMMENTS
(1)	Transfer to AM operations.	RESUME	
(2)	Load dynamic ship file into AM.	EMY	Turn off busy bits.
		LDR	Set up S, R, N.
		RBL	Load words into AM.
(3)	Load SOA into the comparand and perform an exact match.	LDR EMC	
(4)	If no responders, exit.	JNR	
(5)	Advance the response store one position and store the address of responders in core (speed).	RDA	T = 1.
(6)	Halt AM.	CLEAR	
(7)	Call subroutine DIST.		
(8)	Divide each distance by its corresponding SOA and create pseudo data file as described in output list.		

SUBROUTINE NAME: TIME

GENERAL PROCESSING PROCEDURE	1604-B INSTRUCTION	COMMENTS
(1) Initial housekeeping	ENI	Data file index.
	ENA	Set compare addressing.
	SAU	
	ENA	
	SAU	
(2) Determine number of words in file.	ENA	End-of-file flag.
	ENQ	Mask for end-of-file.
	MEQ	Search for end-of-file.
	SLJ	Error.
	ENA	Determine number of words and save.
	SUB	
	INA	
	SAU	
	ENI	
(3) Initialize pseudo file.	LDA	Logical name to data file.
	STA	Increment file index.
	INI	TIME to data file.
	ENA	Increment file index.
	STA	
	INI	
(4) Find logical name and determine number of words in record.	ENA	Logical name flag.
	ENQ	Logical name mask.
	MEQ	Search for logical name.
	SLJ	Determine number of words in record and save.
	ENA	
	SUB	
	INA	
	ENI	
(5) Search for match on attribute.	ENA	
	EQS	
	SLJ	
(6) Determine distance.	LDA	Call subroutine DIST.
	SLJ	
	SAU	
(7) Determine time and place in pseudo data file.	ENA	Dividend to A and Q.
	ENQ	Distance/Speed.
	DVI	Result to data file.
	STA	Data file index.
	INI	
	SLJ	

SUBROUTINE NAME: VLLU

PURPOSE: To perform a variable length lookup.

MAJOR PERFORMANCE LOCATION: Associative memory.

DATA INPUT:

(1) A variable length word along with its associated mask.
(2) Data stored in the AM.

OUTPUT LIST: The 1604-B addresses which contain the word being searched.

CALLING SEQUENCE: VLLU, n, w, m, a

where:

n = no. of computer words in the word being searched.
w = the variable length word.
m = its associated mask.
a = where the output is to be stored.

SUBROUTINE NAME: V

GENERAL PROCESSING PROCEDURE		GAP INSTRUCTION	COMMENTS
(1)	Initialization.	LDI	Clear index to 0
		EMC	Set D buffer to all "1" s } Micr
		EMC	Set E buffer to all "1" s } Oper n
(2)	Do an exact match search, AND results, shift D and E buffers.	LDR	Set up S, R, N.
		EMC	Set T = 1, Z = 1.
(3)	If no responders, exit.	JNR	
(4)	Perform steps (2) and (3), n-1 times.	ICI JIL	Increment index and compare w n, number of words.
(5)	Store address of responders.	RDA	Store addresses in 1604-B locat s starting with a.
(6)	Halt AM.	CLEAR	

Unclassified
Security Classification

DOCUMENT CONTROL DATA - ..&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Auerbach Corp
Philadelphia, Pa.

2a. REPORT SECURITY CLASSIFICATION
Uncl

2b. GROUP

3. REPORT TITLE
Analysis of Small Associative Memories for Data Storage and Retrieval Systems

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Final Report, October 1964 to September 1965

5. AUTHOR(S) *(Last name, first name, initial)*
Green, Robert S., Mr.
Minker, Jack, Dr.
Shindle, Warren, E., Mr.

6. REPORT DATE
July 1966

7a. TOTAL NO. OF PAGES
614 (124 + 490)

7b. NO. OF REFS
315

8a. CONTRACT OR GRANT NO. AF30(602)-3564

b. PROJECT NO. 4594

c. Task 459402

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
1231-TR2

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
RADC-TR-65-397, (Vol I, Vol II)

10. AVAILABILITY/LIMITATION NOTICES
This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC(EMLI),GAFB,N.Y. 13440.

11. SUPPLEMENTARY NOTES
N/A

12. SPONSORING MILITARY ACTIVITY
Rome Air Development Center (EMIID)
Griffiss AFB,N.Y. 13440

13. ABSTRACT

The objective of this effort was to determine the effect of associative memories which are realizable today to aid in processing formatted record problems. The evaluation consisted of a comparison between the CDC 1604B and the CDC 1604B-Associative Memory to process the same problem. The Goodyear Aerospace Corp associative memory was used to establish state-of-the-art in associative memories, however, other associative memory designs were investigated.

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Associative Memories						
Content - Addressable Memories						
Comparison Evaluation						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through

____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through

____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through

____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.

Unclassified

Security Classification