7-1972

Advanced Computer Program Models: A Talking Textbook Based on Three Languages

John Thomas Gardner
Walden University

Follow this and additional works at: https://scholarworks.waldenu.edu/dissertations

Part of the Education Commons

This Dissertation is brought to you for free and open access by the Walden Dissertations and Doctoral Studies Collection at ScholarWorks. It has been accepted for inclusion in Walden Dissertations and Doctoral Studies by an authorized administrator of ScholarWorks. For more information, please contact ScholarWorks@waldenu.edu.
ADVANCED COMPUTER PROGRAM MODELS
A TALKING TEXTBOOK
BASED ON THREE LANGUAGES

by

John Thomas Gardner
B.M.E. Kansas State College, 1951
M.S. Kansas State University, 1957

Robert W. Swanson, Ph.D., Advisor
Director of Computer Center
California State University
San Diego

A Project Submitted in Partial Fulfillment of
The Requirements for the Degree of
Doctor of Philosophy

Walden University
July, 1972
AN ABSTRACT OF A DISSERTATION

By

John Thomas Gardner

B.M.E., Kansas State College, 1951
M.S., Kansas State University, 1957

A Thesis Submitted in Partial Fulfillment of
The Requirements for the Degree of
Doctor of Philosophy

Walden University
July, 1972
The purpose of this dissertation was to develop a learning instrument, to be used by programmers preparing for the Data Processing Management Association Test as a self study book, or by college business programming and computer science students who have completed a course in data processing and a course in programming a higher level language.

The mathematical ability requirement was minimized by developing the algorithms in parallel with the programs.

The learner should experience emphasis in the following areas:

1. The type of activities required to pass the DPMA test (the programming part)
2. Data Structures
3. Fortran (at the level of the DPMA test)
4. RPG (at the level of the DPMA test)
5. Flow chart reading and writing

Fortran and RPG (Report Program Generator) languages were used, since their proficiency is required for the DPMA test; however a subset of IBM Basic Assembler language was used, because the author believed that a person who is more than superficially interested in computers should demonstrate a proficiency with a machine language.

An important part of this method of presentation are the cassette recordings which allow the learner to progress outside the classroom. The recordings plus the hard copy of the actual programs, diminished in size, give the learner material which he can move to any location and study without the presence of the instructor.

Organization

The book is divided into four main parts:
1. Introduction (containing the Swanson Systems Study)
2. Data Structures
3. Fortran
4. RPG

The Introduction contains the Swanson Study which provides a model to answer the question "WHY" for much of the rest of the book.

Section II, Data Structures, includes many of the topics included in the recommendations of the Curriculum Committee on Computer Science of the Association of Computing Machinery.

Section III, Fortran, is designed to prepare the advanced programmer for the DPMA test, but can be used to develop programs in other languages by using the fortran programs presented here as models. This section does contain some BAL programs.

Section IV, RPG, provides models which will aid in the preparation for the DPMA test. The BAL models used here were selected to match the same topic, Invoicing, as the RPG program.

Experimental Results

The experimental objective was to determine to what extent, if any, Advanced Computer Program Models affected the attitudes towards and proficiencies in programming of college level students in San Diego County.

The original design of the experiment was a completely crossed three by four factorial experiment: three level (novice, beginner and expert) versus four types of computer instructions: (1) standard classroom, (2) no classroom, only the talking textbook material, (3) classroom
and material, (4) open study with no guidance from instructor or experimental material.

The observations are individual students' different scores with respect to a standard examination given in January at the start of the school semester, and the same examination given in May at the end of the semester. Classes were held for one night a week for a total of eighteen meetings. One section consisted of students learning computer programming. The other sections consisted, with one exception, of programmers who had some experience with programming. In this section the objective was to develop techniques for data structures.

There were several factors which prevented us from fulfilling the requirements of the originally completed cross factorial experiment:

1. We were dependent on the availability of computer equipment.
2. We could not control the composition of the programming classes so that one class was a mixed grade level and the other class was not mixed.
3. We were dependent upon volunteers and had no control over the number of programming sections assigned to these volunteers.

To measure the affect on proficiency, we used the CTSS test data bank maintained in the California State University, San Diego. To measure the affect on attitude, we used the Aiken Devised Programming Attitude Scale, which is a Likert-type scale, developed by Louis Aiken, professor of psychology, at Guilford College. They were administered once within a week period between January 15th and January 21st, and
the second time between May 15th to May 22, 1972. The tests were administered by the instructors to their own classes. The grading and scoring was performed by the staff at San Diego University.

A statistical analysis was performed with the assistance of the computer facilities of San Diego State University. Biomed Statistical Program, BMD-V, was used in one analysis. The remaining analyses and data summaries were made using the statistical program library (S.T.L.) available at the University.

Conclusions

The conclusions derived from the testing and experiments were:

Novices scored significantly higher with classroom instruction with little difference in text used. The absence of significance in the effects of the classroom experience and text on student proficiency and attitude in the other groups was contrary to expectations. The curriculum in question had been developed for a different approach for the subject. At the end of the experiment, the instructors were of the opinion that for a more effective utilization of the material, it would be necessary to adapt the text to the curriculum.

Whether or not a curriculum especially adapted to this experiment would show any changes in student proficiency or attitude is another question for objective evaluation.

In any case, it is clear that the function of this text, as it was coordinated for this experiment, did not provide any reasonable improvement in the overall student proficiency or attitude beyond the variations associated with students, teachers and experience level.
PREFACE

This book is a learning instrument, to be used by programmers preparing for the Data Processing Management Association Test as a self study book, or by college business programming and computer science students who have completed a course in data processing and a course in programming a higher level language.

The mathematical ability requirement of the student has been minimized, since the algorithms are developed in parallel with the programs.

The learner will experience emphasis in the following areas:

1. The type of activities required to pass the DPMA test, (the programming part)
2. Data Structures
3. RPG (at the level of the DPMA test)
4. Fortran (at the level of the DPMA test)
5. Flow chart reading and writing

Fortran and RPG (Report Program Generator) languages were used since their proficiency is required for the DPMA test; however a subset of IBM BAL language was used because I believe that the person who is more than superficially interested in computers should be able to demonstrate a proficiency with a machine language.

The cassette recordings are an integral part of the presentation, since they allow the reader to progress outside the classroom. The recordings, plus the hard copy of actual programs, diminished in size, give the learner material which he can move to any location and study without the presence of the instructor.
ORGANIZATION

The book is divided into four main parts:

1. Introduction (containing the Swanson System Study)
2. Data Structures
3. Fortran
4. RPG

Section I, the Introduction, contains the Swanson System Study, which provides a model to answer the question "WHY" for much of the rest of the book.

Section II, Data Structures, includes many of the topics included in the recommendations of the Curriculum Committee on Computer Science of the Association of Computing Machinery.

Section III, Fortran, is designed to prepare the advanced programmer for the DPMA test, but can be used to develop programs in other languages by using the Fortran programs presented here as models. This section does contain some Basic Assembler programs.

Section IV, RPG, provides models and information which will aid in the preparation for the DPMA test. The Basic Assembler program presented here is used as a comparison for the invoicing program, written in RPG.
HOW TO USE THIS BOOK

The programmer reviewing for the DPMA test can study the Swanson Study, use the cassettes, review the Fortran Section (paying particular attention to the first half), and review the RPG Section. This can be done in six weeks without an instructor.

The college student can make a semester, or a year study, with or without an instructor. The type of computer is incidental, since the tape cassettes analyze each BAL program. In fact a computer isn’t needed (however it does work as a motivating force). This allows several different languages to be programmed at the same time during a semester.

An instructor who uses this book need not know the Basic Assembler Language, because the tapes will assist him.

The long programs were diminished in size intentionally so that a learner could manipulate them with ease. It is sometimes better for analysis, if the learner copies certain sections manually, if he wants to carry on a concentrated study.

The book starts with a real systems study, which uses a fictitious name. The study was made by the firm of Swanson Associates, (real name), and the corporation in the study is the Marine Division of the Oceano-Graphic Corporation (fictitious name).

Why use a systems study to start a programming text? The answer to this question is that the study gives meaning to the whole text, and serves as a vehicle for the text. The programmer reviewing for the DPMA test might have lost view of the whole corporation, he might have forgotten some of his systems work he had a few years ago,
and might not have a model of a systems study to review before his test.

The college student might have a good background in accounting, but it will add to his ability if he can work with a model that develops a management information system, which includes the whole or at least a part of the corporation.
ACKNOWLEDGEMENTS

Information and illustrative materials have been used from publications copyrighted by the International Business Machines Corporation.

Direct quotations appear as itemized below:

1. C-26-3600 RPG Programs Invoicing and Pre-Billing Calculation with Inventory Control

2. C-33-6000 RPG Disk Examples

The Systems Study was adapted from a real systems study by Dr. R. L. Swanson and Associates.

Help from many sources has contributed to this book. To my students and teachers, named and unnamed, I extend my deepest gratitude. I also express my thanks to Mr. Wayne Langley, Mr. Richard Schenck, Mr. Lawrence Huster, Mr. Herman Dailey, Mr. Roy Towne and Mrs. Ethel Castle, who taught, shared knowledge and provided aid in working various programs.

Special thanks go to Dr. Robert Swanson, who patiently guided me through the dissertation and provided the systems study upon which the book is based.

Most of all, I wish to express my deepest appreciation to my wife, Patty, and seven children, Clorinda, Patrice, Deborah, Curtis, Melisa, Michele and Melanie, who were, I am sure, very pleased when "the computer book" was finished.
# TABLE OF CONTENTS

## PART I  INTRODUCTION

1. Swanson System Study .................................................. 2
2. Construction Program .................................................. 47
3. AVCC Program .................................................................. 72

## PART II  DATA STRUCTURES ..............................................

2.1 Introduction .................................................................... 79
2.2 Linear Lists ..................................................................... 87
   2.2.1 Stacks, Deques and Queues ..................................... 88
   2.2.2 Linked Lists .......................................................... 91
2.3 Trees .............................................................................. 101
   2.3.1 Hierarchical Ring Structure .................................... 104
   2.3.2 Traversing Trees—Preorder and Post Order .............. 107
2.3.3 Directories .................................................................. 115
   2.3.3.1 Structured Tree Directory .................................. 115
   2.3.3.2 Random Ordered Directories ................................ 116
2.4 Files .............................................................................. 117
   2.4.1 Sequential Files ...................................................... 119
   2.4.2 Inverted Files ......................................................... 122
   2.4.2.1 Creation, Maintenance and Deletion of Keys ....... 123
   2.4.2.2 Logical Operators .............................................. 125
   2.4.2.2.1 Logical Operators OR ................................. 126
   2.4.2.2.2 Logical Operator "AND" ................................. 128
   2.4.2.2.3 Logical Operator "AH D" ................................. 130
2.4.3 Multilist File ............................................................. 132
   2.4.3.1 Deletion of Records and Addition of Keys ........... 133
2.4.4 Cellular Multilist File .................................................. 135
2.5 Sorting ............................................................................ 136
   2.5.1 Sorting Methods ...................................................... 136
   2.5.1.1 Key Sort .......................................................... 136
   2.5.1.2 Detached Key Sort ............................................ 137
   2.5.1.3 Non Detached Key Sort ..................................... 137
   2.5.2 Radix Sort ............................................................. 139
   2.5.3 Quicksort ............................................................... 140
   2.5.4 Topological Sort ...................................................... 141
   2.5.5 Quadratic Sort ....................................................... 142
   2.5.6 Calculated Address Sort ........................................ 143
   2.5.7 Sorting by Insertion .............................................. 144
   2.5.8 Interchange Sort .................................................... 146
2.6 Merging .......................................................................... 148
   2.6.1 Two Way Merge ..................................................... 148
   2.6.1.2 Merging Ordered Files with Subfiles ................. 150
   2.6.1.2.1 Merging Step Down ...................................... 154
Table of Contents (con't.)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7 Search</td>
<td>158</td>
</tr>
<tr>
<td>2.7.1 Linear Search</td>
<td>158</td>
</tr>
<tr>
<td>2.7.2 Binary Search</td>
<td>158</td>
</tr>
<tr>
<td>2.8 Dynamic Storage Allocation</td>
<td>163</td>
</tr>
<tr>
<td>2.9 Virtual Memory</td>
<td>166</td>
</tr>
<tr>
<td>PART III FORTRAN</td>
<td>172</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>173</td>
</tr>
<tr>
<td>2. Summary of Statements that Exist in Fortran I, II and IV</td>
<td>175</td>
</tr>
<tr>
<td>3. Expression Formation</td>
<td>178</td>
</tr>
<tr>
<td>4. Hierarchy of Operation</td>
<td>178</td>
</tr>
<tr>
<td>5. Nested Iterative Input and Output</td>
<td>179</td>
</tr>
<tr>
<td>6. Fortran Shortcuts</td>
<td>182</td>
</tr>
<tr>
<td>7. Buffering with Fortran</td>
<td>183</td>
</tr>
<tr>
<td>8. Syntax</td>
<td>183</td>
</tr>
<tr>
<td>9. Array Search</td>
<td>184</td>
</tr>
<tr>
<td>10. Sort Ascending and Descending</td>
<td>185</td>
</tr>
<tr>
<td>11. Installment Note Program</td>
<td>187</td>
</tr>
<tr>
<td>12. Statistics</td>
<td>188</td>
</tr>
<tr>
<td>12.1 BAL Statistics Program—STAT</td>
<td>189</td>
</tr>
<tr>
<td>13. Selected Fortran Programs in the Form of Answers to Questions</td>
<td>196</td>
</tr>
<tr>
<td>13.1 Compound Interest Rate</td>
<td>197</td>
</tr>
<tr>
<td>13.2 Installment Note</td>
<td>198</td>
</tr>
<tr>
<td>13.3 Integration by Simpsonas Rule</td>
<td>199</td>
</tr>
<tr>
<td>13.4 Quadratic Equation</td>
<td>200</td>
</tr>
<tr>
<td>13.5 Linear Equation</td>
<td>201</td>
</tr>
<tr>
<td>13.6 Compound Go To</td>
<td>202</td>
</tr>
<tr>
<td>13.7 The Derivative</td>
<td>203</td>
</tr>
<tr>
<td>13.8 Evaluate a Polynomial</td>
<td>204</td>
</tr>
<tr>
<td>13.9 Functions</td>
<td>205</td>
</tr>
<tr>
<td>14. Arithmetic Questions and Answers</td>
<td>206</td>
</tr>
<tr>
<td>PART IV RPG</td>
<td>219</td>
</tr>
<tr>
<td>1. Pre-Billing Calculation with Inventory Control</td>
<td>222</td>
</tr>
<tr>
<td>2. Customer Invoicing RPG</td>
<td>259</td>
</tr>
<tr>
<td>3. Customer Invoicing BAL</td>
<td>280</td>
</tr>
<tr>
<td>4. RPG Coding for Disk Operations</td>
<td>298</td>
</tr>
<tr>
<td>GLOSSARY</td>
<td>337</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>355</td>
</tr>
<tr>
<td>INDEX</td>
<td>370</td>
</tr>
</tbody>
</table>
SECTION I

INTRODUCTION
THE SWANSON SYSTEMS STUDY

The objective of this section is to present a complete model of a systems study, and the presentation of the files serve as models to be programmed at the end of the book.

Since so many elementary programming books start with explanations of computers and use only short examples, it was felt that an advanced book should contain a system study and complete models of programs when possible.

It is suggested that the reader return again and again to the charts of this study so he can develop a feeling for the functional groupings of this real corporation.
PRODUCTION SYSTEM PROPOSAL

for the

MARINE DIVISION

of

OCEANO-GRAPHIC CORPORATION

by

Dr. Daryl G. Mitton

Dr. Robert W. Swanson

1.0 ABSTRACT

This is a proposal of a system design for the Marine division of the Oceano-Graphic Corporation. We believe its unique features will provide a multitude of advantages for the firm including:

a. A planning and control capability which cannot even be approached by your current system and which, we feel, has not been attained for any similar sized company in the past.

b. A capability of response through information accessibility for all tactical and strategic situations.

c. A reduction in both direct and indirect production costs— at present or increased production levels.

d. A competitive advantage in the ability to obtain proper proposal information from a technical, cost, and delivery standpoint.
e. A reduction in inventory and inventory costs (we feel that under existing output levels, the present inventory level of $85,000 can be cut to $20,000).

f. The availability of analyses of consequences of alternative decisions.

g. The many sundry advantages brought about by good systems design, such as more complete and improved cost information, the signaling of impending operational difficulties, the reduction of operation cycle time, the improvement of customer service and feedback, the determination of future equipment and manpower needs, and the preparation of most essential paperwork.

The strength of the system is the ability to obtain needed information easily. This strength provides a strong motivation to keep the system operative. No operational system, regardless of its detailed design, will function properly unless it is used as its design intended it to be used. Most systems are designed in such a way that their intended function is obstructed in some way to satisfy organizationally local needs. Therefore, most systems fall short of their intended goals.

At present, the Marine Division of the Oceano-Graphic Corporation has a very detailed and elaborate system for operational planning and control. It does not function. Its literal atrophy is caused by an informal realization at all levels within the division that the struggle to supply input and the difficulty of receiving output are not worth the rewards obtainable by the functioning system.

This is no direct criticism of the Marine Division system, for what is symptomatic here is found in most organizations: the super-
imposition of an informal "make-do" system which operates on "get by" and exists to accommodate certain paper work demands that may or may not have any real relevance to the effective running of an organization. In fact, it is not uncommon for the resulting working system to be actively dysfunctional, particularly in the long run.

Our study first confirmed the existence of a non-working system. We next submitted preliminary designs of two possible alternative systems which would provide a greater degree of automaticity in speeding input and spilling output. The first was a somewhat conventional approach, an integrated data processing system, which would mechanize the various subsystems (a subsystem at a time) that impacted on the production function—production, purchasing, accounting, engineering, etc. Each system, being separate to orient its particular output demands, would, nonetheless, be updated by information fed into any of the subsystems. Output would have a specialist orientation—not suited to generalship. The data files would be redundant and the system would never be totally current. We admit to editorializing against this system in favor of a second approach, which the Marine Division decided to pursue.

The system devised is built around the concept of a single flow of information. Essential to this system is a common data base. Information input is, therefore, immediately triggered to update purchasing, production, engineering, etc. Information is always current. No output is made unless called for. Information release is on a "need to know" basis rather than a "nice to know" basis, and it is in an interpretable form to serve as a decision base. The ability to obtain needed information very easily serves as the strongest motivation to supply essential input. In spite of the universality of the system, we have
devised a way to introduce it in modules, which will enable faster realization from the system as it is programmed and introduced, and which provides for current debugging of the programming.

We recommend an IBM 000/000 computer as the equipment best suited for immediate and future needs of the Marine Division. We estimate its first-year costs at $00,000 ($0,000 per month for ten months) to program and debug the system recommended. Current Marine Division personnel would be able to man the program, so no additional cost would be incurred here. It is unfortunate, in a report such as this, that we cannot include as a cost the price of not making the change, for we feel, if the Marine Division is to be a part of the growth of oceanographic explosion that is sure to come, these changes must be made.

What is the cost of losing a position of leadership in oceanographic instrumentation? What is the cost of losing a contract because of poor performance? What is the cost of slipping ever so slightly behind the competitor? There are costs that do not show in the profit and loss statement directly, but their costs are just as real as those that are displayed.

1.1 PLAN OF REPORT

The report is presented in five parts, starting with a discussion of the objectives and limitations. This is followed by an explanation of the methods used and alternatives available to us in conducting the study and design. The next section of the report is devoted to a summary analysis of the existing system and what we considered to be the major limitations for continued successful use of the system as it currently exists. Beginning in Section 5.0, we present the proposed single
information flow system, complete with the details of the proposed filing system. In Section 5.5, we further specify the benefits that can accrue from the employment of such an approach to the production information system. Section 6.0 of this report deals with our estimated cost and schedule of implementation.

2.0 OBJECTIVES

The objective of this study is to analyze existing procedures and conditions of operations prevalent in the Marine Division of the Oceano-Graphic Corporation to determine the feasibility of use and choice between available methods of operations of automatic data processing systems.

A part of this objective has been to develop an operating system that would provide the management of this division with insight and control of the production function, insight being defined as access to the facts in an orderly manner with an insured degree of accuracy.

A second function of the objective was to develop the state of automation to provide individual response to the system.

The functional criteria for measuring how well these objectives can be obtained are:

a. Reduction of the number of manual checking functions by one-half (i.e., routine decisions and paperwork).
b. The increase of the division's profit to thirty percent prior to taxes.
c. Reduction of the total direct and indirect cost of production by twenty percent.
d. Reduction of the response time to a customer's order to thirty days.
2.1 SCOPE

There are many limitations to a report of this nature. Time only permits the minimum of attention to the existing disciplines and methods of management information systems. The chief center of interest here is to determine the prospective utilization or applicability of present systems. Further, since many management decision factors are prefaced upon future events not totally definable now, no effort has been made to define the future needs of Oceano-Graphic other than what is needed now. These decisions must be reserved for the future as the technology evolves; however, the provisions for growth were considered and incorporated as a function of the study. When we consider the peculiarly sensitive nature of a company's investment in computers—with its potential for major impact on its position in competition—judgments on how, where and when—the investment should be made to assume special importance. As a result of advances in technology, the computer can now play a more central role in corporate planning and operations. Since present criteria for allocating resources to ADP are based on outdated data processing operations or on rules of thumb, they do not take into account the ability of computers to contribute to profits and/or cut operating costs outside of ADP. Clearly, it would be more appropriate to evaluate the computer investment in terms of its contribution to the entire management process.

We recognize how difficult a task this will be. There can be no simple rules of thumb for analyzing Oceano-Graphic's ADP expenditures. These expenditures must be geared not to the company's size or competitor's spending, but to the benefits to be derived in each specific case. Since these benefits often accrue from the value of the information to
be supplied by the system, as well as from the added efficiencies and direct savings in data processing operations, they are elusive and extremely difficult to measure.

It should further be expressed as a function of the scope of this study that we, as consultants, are not in the business of selling computer hardware. Our task is to evaluate the present system and recommend changes as they relate to the objectives of the study. If the results of this study are implemented, the choice of equipment would be the decision of the Oceano-Graphic Corporation.

3.0 BACKGROUND

Our approach to this particular study was to review the present system as it is currently operating. This review included those documents and procedures that are defined in the systems and procedure manuals and a complete survey of the actual operations and procedures as interpreted by those performing the functions. There always exists a certain dichotomy in all enterprises. There are the procedures set by the workers and lower levels of management and the procedures formally defined by top management. Rarely are the interpretations identical. Suborganizations with organizations adopt symptoms to satisfy the needs of their organizational area. They add to and ignore the system as they see it and make it tolerable and workable from their point of view. Therefore, there exists a system working the way the lower echelons believe it has to work. It is necessary to review both systems.

Each of the activities was reviewed from the perspective of how well it contributed to the production of the end product and how well it fit into the function of management control. Instead of starting
with the costs of processing data, we started with the positive value of the information produced by the processing. The benefits do not stem directly from the fact that given data are processed; they stem from the results of data processing. The value of information from data processing is not what it costs to obtain but, rather, what it can do for management. Costs have their place, but in a different part of the equation.

This, the first effort becomes one of analysis of the intended system. It is an effort to establish a system based upon the total objectives of the company. Recognizing the profitability of rapid growth in the oceanographic field and the limitations of time and cost in designing and implementing a total system, we feel there are two feasible alternatives to choose from—integrated data processing or single information flow.

The integrated data processing approach is an attractive alternative from several aspects. First of all, it is the conventional approach. Most commitments in the field of data processing take this approach. It is a step-by-step approach. You can merchandise a subsystem at a time, thereby getting a system on the computer, gaining some utilization and economic value out of the system without the total system being designed. In this case, production could be put into operation first, followed by a purchasing system, then accounting, etc. All of the subsystems would be designed so that information from one system would be used to update other subsystems, thus integrating all the systems into a single integrated whole. Each of the subsystems would produce the desired reports. In the technical jargon, we refer to it as an "output oriented system."

The single information flow concept developed from some of the
apparent limitations of the integrated data processing (IDP) approach. First, consider the IDP method each subsystem being linked to the other subsystems. This means each subsystem must be updated without respect to the volume of transactions affecting it. Secondly, each subsystem must, by its nature of independence, support separate filing systems, much of which is redundant data. This requirement for processing and redundant files necessitates considerably more machine capacity than would be required if they could be eliminated. A second deficiency arises in that the system is never totally current by the nature of the interlacing of subsystems. Thus, management in each of the separate functions always sees different data creating a continuous point of contention.

This system is designed whereby the source information in any department updates a single information file. For example, an engineering change would update the purchasing files and production files, as well as the engineering files at the time of entry. Conceptually, they would be a single file. In this mode, we could eliminate a considerable amount of processing. In the technical sense, the system would become input oriented. The results would then be a system containing the same information bank, but requiring less equipment and more current data.

When presented with these alternatives, the management of the Marine Division chose to pursue development in the direction of the single information flow concept. The design of the system was then undertaken with the intent of developing this concept on a limited scope, primarily directed to production, inventory control, material control, and purchasing functions.
3.1 PROCEDURAL ANALYSIS

The benefits accruing from any management information system, whether computerized or not, may be seen to fall into three categories:

a. Cost savings—i.e., savings in data processing cost because of reductions in clerical work force and other changes.

b. Operational gains—i.e., efficiencies in corporate operations resulting from the application by managers of information received through the system (for instance, data on inventory reductions and faster production).

c. Intangible benefits—i.e., improvement in customer service, corporate planning and forecasting, the ability to sustain growth, and other advantages which may not be present without the system but which depend upon management's astuteness in using it.

Thus, the functions for measuring the value of the present system were categorized into these three areas in an attempt to establish a baseline.

It is also necessary at this point to differentiate between the business system and the information system of the enterprise. For the most part, the business system has its main function of converting the raw materials, through a process of manufacturing, testing, and assembling, into oceanographic measuring instruments. The management information system, on the other hand, translates from the environment and from within its own components as its inputs. It stores this information and associates it with previously stored information in order to provide a frame of reference for the next vital step, that of making a decision.
Defined in this manner, the analysis of the present system and design of the proposed system were evaluated against the following premise. Management control is a function of measuring how well a task is being accomplished as compared to what was planned. This is pictorially described as:

4.0 THE EXISTING SYSTEM

As previously stated in the procedure of analysis section of this report, the existing system was evaluated primarily from the following points of view:

a. What are the objectives or premise for operations?

b. How was the premise put into operation, i.e., how did the planning take place?

c. In what manner was the plan implemented?

d. What was the measurement for determining how well the plan was being accomplished?

e. What was the medium for feedback and how did it affect the process?

The first step was to define the processing system as it currently exists. EXHIBIT "E1" describes the system by means of a PERT type network as it was described by the employees within the system.

As we appraise the system from an overall standpoint, we find
Detail of Production

Figure SWAN 1
the information is usually defined in two documents, the MJO and contract. This information is then combined with the forecast of sales to plan the task. The planning task for inventory MJO's is completed and issued in the form of a milestone schedule, which is filed and never distributed. As a matter of interest, during the period of this investigation, much dissatisfaction was expressed by those making this schedule and they had discontinued its process and were in the process of designing a new one. No where in the organization was any other reference made of the milestone schedule nor did anyone seem to miss it. This was somewhat overcome by the authorization for work document. This document does give the information as to what is to be done and some limiting criteria, such as budget and when due.

In pursuing the authorization for work as the planning and control mechanism, we found its primary use was to provide the major task definition and a charge number by which to accumulate costs. It contains no provision for stating how the job is to be done and if there is a change in the plan. For instance, there is no revising of this document to reflect the change. Hence, the document doesn't really provide the planning function nor the feedback medium for control.

To further pursue the process of planning, we found the foreman in each of the departments was familiar with what had to be accomplished, based upon past experience, and assigned the work accordingly. This work assignment was related to their area only and reflected none of the interface requirements of the other subsystems. Thus, the product test plans were in no way coordinated with the machine shop. Again, there was a complete lack of any formal feedback mechanism. It would appear to an outsider that the system primarily reacted to crises and was not
The master of manufacturing
checks the MHD from the financial
department. He also secures
a copy of the contract to determine
what must be purchased. He also
refers to the forecast of sales.

With this information he
personally develops a workable chart.
At the same time there is some
research of all sources to get and
a request for the drawings. If these
elements have any effect the
chart is then revised and then filed.

The MHD also triggers a
material take-off in the design department.
After this operation is performed
blueprints are sent to the machine
shop. Machine shops determine
material needs; orders same;
plan job and class the processing.

A reference to schedule
prepared earlier Planner did on
planning in view of normal function
of that piece of paper.

Standard parts were issued
in kit form along with a Bill of
Materials maintained on file. The
precise origin of the Bill of
Materials could not be made
traceable to the engineering
change but was assumed to
be correct.

Now standard parts were then
requisitioned in purchasing and went
through the purchasing cycle.

Figure SWAN 2
Source Documents

Customer: OCEANIC GRAPHIC MARINE DIVISION
Date: 00/00/00

<table>
<thead>
<tr>
<th>Description</th>
<th>Number of copies</th>
<th>Which copy/How many</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST FOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MASTER USER</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:

1 COPY: THE ORIGINAL, LATER REPLOYED TO 16 COPIES AFTER APPROVAL.
VOLUME = 450/yr. 15/-pen/one time.

Processing Frequency:
D=Daily  M=Monthly
W=Weekly  Q=Quarterly
B=Bi-Weekly SA=Semi-Annually
S=Semi-Monthly A=Annual
### Report Distribution and Use

**Key:**
- B = Bi-Weekly
- M = Monthly
- A = Annually

<table>
<thead>
<tr>
<th>Report Title</th>
<th>B</th>
<th>M</th>
<th>J</th>
<th>O</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Order</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- Work orders cut on an as needed basis. Approximately 450 per year with an average of 90 being open at one time. Cost/Value study not made as a direct cost.
- MOJ triggers work order authorization. One is redundant.

- B = Bi-Weekly
- M = Monthly
- A = Annually

- Report Distribution (Dept., Function, Person, Etc.)
- Report Distribution (Dept., Function, Person, Etc.)
- Report Distribution (Dept., Function, Person, Etc.)
- Report Distribution (Dept., Function, Person, Etc.)
<table>
<thead>
<tr>
<th>Report Title</th>
<th>Value Narrative</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>F2 1</strong> FORECAST OF SALES</td>
<td>THE forecast is of value to management, but is of little value to Production planning. No commitment to production can be made on the basis of a forecast nor is there any motivation factor between the issue of the work authorization and the sales forecast. There are no statistical procedures computed to establish any measure of universality. In my discussions with respect to production planning under present system, the report has only an immediate value, if any.</td>
</tr>
<tr>
<td><strong>F2 2</strong> MOD &amp; WORK ORDER AUTHORIZATION</td>
<td>Basically these documents, the MOD is the only documented evidence of planning that appears to tie all operations together. Document contains customer identification, funding, budget, and 2nd approval. The authorizations part does identify the major functions to be accomplished. However, it is department oriented and not process oriented. Budget is not related to time nor work station, may or may not reflect estimate.</td>
</tr>
</tbody>
</table>

*Consider at least (1) Earning on freed capital, (2) Yield on inventory reductions, (3) Benefits, (4) Equipment, (5) Disposable costs, (6) Profit on increased sales.*

**Figure SWAN 5**
# Request For Master Job Order

**Initiator:** [Name]

**Project Mgr:** [Name]

**Project Eng:** [Name]

**Customer Name/Address/Job Title:** [Details]

### Type of Project:
- Contract/P.O.
- Mfg. for Inventory
- Construction of Asset
- Testing
- Basic or Applied Research
- D.C. Repair

### Contract Funding

<table>
<thead>
<tr>
<th>Cost</th>
<th>Fee or Profit</th>
<th>Total</th>
</tr>
</thead>
</table>

### Contractual

| Completion Date

### Purchase Orders or Subcontracts may be placed against this MJ on or after

### Contracts Approval

<table>
<thead>
<tr>
<th>Engr. O/H %</th>
<th>Mfg. O/H %</th>
<th>C&amp;A %</th>
</tr>
</thead>
</table>

### Approved:

**Area of Budgets:** [Name]

**Plant/Project Manager:** [Name]

**Vice President:** [Name]

**Pres/Exec. Vice President:** [Name]

---

### 2. MASTER JOB ORDER

**MJ No.:** [Number]

**Change No.:** [Number]

**Cross Reference:** [Details]

**Expiration Date:** [Date]

**Ship to:**

**Effective Date:** [Date]

**Billing Instructions:**

- Taxable [ ]
- Non-Taxable [ ]

**F.O.D. Point:** [Details]

**Terms:** [Details]

**Net:** [Details]

**Days:** [Details]

**Via:** [Details]

**Reports and Other Requirements Including QC:**

- [ ] See attached

**Scheduled Completion Date:** [Date]

**Scheduled Shipping Date:** [Date]

**Distribution:**

- [ ] Budgets
- [ ] File
- [ ] Mater.
- [ ] Accrual
- [ ] Eng.
- [ ] Mfg. Control
- [ ] R&D
- [ ] Pur.
- [ ] Ship
- [ ] E&D
- [ ] S.P.T.
- [ ] QC
- [ ] Doc. Control
- [ ] Mfg.

**Approved:**

**Vice President, Administration:** [Name]

---

**Figure SWAN 6**
Authorization For Work


to: ____________________________

From: ____________________________

Customer: ____________________________

Project Title: ____________________________

Start Date: ____________________________

Due Date ____________________________

Budget ____________ ____________

Labor ____________________________

Mat'l ____________________________

PROJECT APPROVALS

Plant Mgr. ____________________________

Dir., Marine Div. ____________________________

Vice President ____________________________

TASK APPROVALS

Project Mgr. ____________________________

19

Figure SWAN 7
## VALUE COMMENT

<table>
<thead>
<tr>
<th>Report Title</th>
<th>Value Narrative</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCTION PLANNING OPERATION</td>
<td>DOCUMENT CONTAINS THE SIGNIFICANT IDENTIFICATION DATA, CONTENT IN SEAM AS A PLANNING DOCUMENT IS USELESS, IN EXCEPTED FUTURE FACTORY OPERATIONS. BEAR NO RESEMBLANCE TO PRESENT OR FUTURE, TOOLING TIME STANDARDS ARE NOT MAINTAINED OR RELEVANT. CHECK INVENTORY CONTROL STATE SHEET PRIOR TO REVISION OF MAKING PLANNING, CONSISTS OF REPRODUCING A COPY FROM A WRITE-ON PLANNER PREVIOUSLY REPRODUCED, OPERATIONAL BEAR NO CORRELATION TO PRESENT PRODUCTION VOLUME OR REQUIREMENTS. DOCUMENT COULD DISAPPEAR AND HAVE NO EFFECT ON THE SYSTEM. IF THIS PROBLEM WERE TO BE ADDRESSED TO FUNCTION CORRECTLY, INVENTORY COST SAVINGS COULD BE REDUCED FROM CURRENT $35,000 LEVEL TO A $20,000 LEVEL EASILY. THIS COULD BE ACCOMPLISHED WITH AN IMPACT TO PRODUCTION REQUIREMENTS. RECOMMEND A BAY WHATEVER NEEDED TYPE OF SYSTEM.</td>
</tr>
</tbody>
</table>

*Consider at least (1) Earning on freed capital, (2) Yield on inventory reductions, (3) Benefits, (4) Equipment, (5) Displaceable costs, (6) Profit on increased sales.

---

Figure SWAN 8
<table>
<thead>
<tr>
<th>Item</th>
<th>Dept.</th>
<th>Manufacturing Operation</th>
<th>Tool No.</th>
<th>Std.</th>
<th>Act</th>
<th>Time</th>
<th>Initial Date</th>
</tr>
</thead>
</table>

Figure SWAN 9
a function of discerning which of many alternatives should be taken.

Unfortunately, it is the empty wagon that makes the most noise.

Because of the lack of a formal flow of the planning function, it became impossible to measure economically the cost of that system and we, therefore, abandoned attempting to establish that of baseline. We also recognize that the Marine Division is basically an intermittent type of operation or job shop—organized about multipurpose machines to perform specific functions. This type of environment normally relies on heavy paperwork in order to control closely its cost and production capacities.

We believe the present management has designed systems that would accomplish the intent of control. However with the dynamics, particularly present in the oceanographic field, any effort to accomplish this manually with a sizable production volume would be futile. First, the cost of manpower, primarily clerical and management, would be economically infeasible. Second, the response time would be of such length as to negate most of the competitive advantages of the product and seriously affect the function of sales.

For the purposes of this report, we have included some of the analytical data used to arrive at the above conclusion. This, of course, is only a sample of the effort and volume of data collected. It is not our intent to burden this report with data that is meaningful only to fellow colleagues in this field. However, some insight should be given the reader as to the logic of the conclusions.

5.0 PROPOSED SYSTEM

Experience has shown that the gathering and dissemination of
information is the manufacturing company's most difficult problem. Information is voluminous, scattered, and often difficult to obtain. In one of the early stages of this study, Mr. Fictitious emphatically stated he wanted a system that would give him "access to the facts."

In addition, if we are to maintain the function of management control, we must provide for a means of planning and measurement that is currently lacking authenticity in the current system. Thus, our system must have a central information system and a framework that will facilitate mechanization.

We are proposing a single flow information system where the information will be accumulated in a production control center and where, literally, one set of books is maintained. This takes the form of records stored on computer disk files, readily accessible to all interested parties at a moment's inquiry. These records are designed to contain as much data as is deemed important to management. The accuracy of the records, also an essential element, will be easier to achieve and maintain, for only one set of archives will now be put to use.

The essential element of this system is the development of a data base that generally covers all the operational information needed to handle this company's business. It will be stored on disk files and, therefore, directly on-line with a computer. Because of this, summary and detail information can be accessed, updated, and retrieved from multiple entry points. Data is stored one way through reference to symbolic record field labels and can be printed in various output formats.

Each system of records is linked in a particular manner. For example, a part number, accessed through the basic record, may lead to
what we call the product structure—a where-used going-to file, or standard routing record or the manufacturing sequence, or an open order status, or an open job summary, or a detail record. In the latter instance, the specific work center in which the job is being performed may even be pinpointed.

5.1 THE SYSTEM OVERVIEW

Drawing No. 1, System Overview, shows the interaction on data flow within the Marine Division organization. The interaction of these events is grouped within seven major areas:

a. Sales analysis
b. Engineering
c. Inventory control and production scheduling
d. Manufacturing facilities
e. Finance
f. Purchasing
g. Sales and distribution

While the data base is designed to process segments of all these areas, it does not contain the additional detail records needed to handle sales analysis, finance, and sales/distribution.

5.2 PRIMARY FLOW OF INFORMATION

The input of information leads from an initial input of customer orders and statistical sales background data to the final shipment of an order. A generalized statement of the system flow is divided into a planning phase and an execution phase.

Planning begins with the preparation and projection of order forecasts. Stock availability and on-order status are screened across
Figure SWAN 10
product inventory records, but family component characteristics of the product line must also be recognized. Product structure on bills of material enter into these decisions.

After a determination of net requirements, an order quantity analysis takes place to ascertain lot sizes and lead times for both purchased and manufactured items.

To-buy items are routed to where items are placed on a purchase requisition. At this point a selection of vendor is made, price and delivery are negotiated, and purchase order is released. Receipt cards and/or a scheduled receipt document may be prepared simultaneously with the purchase order and forwarded to the inspection-receiving area of the plant. An open purchase order record is now initiated for follow-up.

To-make items are routed to production planning for assembly and fabrication. Some similarity exists within these two units. An assembly order is generated for the assembly area, a shop order for the fabrication area. Material requisitions and job tickets accompany both documents. Three basic types of records (standard routing, work center load, and open job order) permit assembly and fabrication to schedule, to load, and to level the line or shop and to release the order paperwork.

Execution begins at the purchasing level with the need for order follow-up and vendor expediting. The vendor ships material, accompanying his shipment with packing lists and an invoice.

Varied execution functions are performed at the assembly and fabrication levels. Orders are dispatched, rescheduled, and expedited between work centers. In the meantime, current production reporting
updates work center and open job order records.

5.3 SYSTEM FLOW

Our proposed system has been designed to fit this basic production model. For ease of implementation, eight subsystems have been developed. This allows for modular programming and should substantially reduce the cost of implementation and speed of accomplishment. The information flow begins from two directions. (See Drawing No. 2, Production Model.) The first path moves from engineering data control, to inventory control, to requirements planning. The mission of the engineering data control subsystem is to organize and maintain basic records. These basic records are what we call the item master file, product structure file, standard routing file, and work center master. The subsystem has the added capacity or capability of retrieving information from the data base. Six retrieval functions are presently developed, three in assembly sequence and three in parts usage sequence.

An inventory control subsystem follows organization of basic records. On-hand inventory, usage history, and on-order fields are used in the item master file so that stock status reports can be generated. Thus, a major objective of this application area is record maintenance and updating inventory. With accessibility to such data, "when to order" and "how much to order" decisions are made.

A second flow line moves from sales forecasting to requirements planning. The sales forecasting subsystem analyzes historical demand data, which may be stored on the item master, to provide requirements planning with a gross finished product forecast plan.

The merger of requirements planning with inventory control now
Production Model

Sales Forecasting

Requirements Planning

Purchasing

Assembly/Quality Planning

Operation Scheduling

Shop Floor Control

Inventory Control

Fabrication/Quality Planning

Operation Scheduling

Shop Floor Control

Engineering Data Control

Production Test/Quality Planning

Operation Scheduling

Shop Floor Control

Data Base Records

Eng. Data Control
- Item Master
- Product Structure
- Standard Routing
- Work Center Master
- Open Job Order
- Order Summary

Inv. Control
- Item Master
- Requirements Planning

Purchasing
- Item Master
- Purchase Order
- Vendor Master
- Open Purchase Order
- Capacity Planning

Operation Scheduling
- Standard Routing
- Work Center Master
- Open Job Order

Shop Floor Control
- Item Master
- Standard Routing
- Product Structure
- Open Job Order

Drawing 165 2

ANS
makes it possible to determine net requirements, projected into time periods and schedule due dates. Product structure records are used at this point to allow breakdown of finished product items into individual components. These are similarly netted and projected into time periods. All of this results in planned orders, destined to each of the four links: purchasing, electronic assembly, fabrication, and product test.

Planned orders to purchasing result in material requisitions being prepared. Through the use of purchase master and vendor master records, a vendor may be selected and a purchase order with receiving documents created so that purchase follow-up can be initiated in the next sequence of events.

Planned orders to the three production areas go to the capacity planning sub-systems, or long-range scheduler. Its purpose is to identify overloads far enough in the future for both facility and manpower planning. After order start date calculations are performed (utilizing standard routing records), consideration is given to plant capacity. The work center master is used for this purpose. Available techniques are then used to level the loads. A work center load report, projected by time period, is one of the key output documents. The operation scheduling accepts orders which have gone through a releasing cycle from capacity planning and schedules the work center within its short-range time span. Dispatching sequences are prepared and analyses made of the loads. Priority rules are set and order completion dates determined.

To the short-range scheduling phase of this subsystem, we have added the control of tools. A tool master record is designed for this function.

Shop floor control is the final subsystem in the flow line. It prepares the shop packets and other factory documentation. It also
constructs the open job summary and operation detail records so that the progress of the work can be reported. Feedback is one of its more important functions so that the system can respond to change.

5.4 STANDARDIZED RECORDS

A set of standardized record layouts is included in this report. These records are designed as the data base to mechanize the application areas we have discussed and lead toward developing a single information flow concept.

These records contain the fields we consider necessary to enable Oceano-Graphic Division to utilize and control their control production output requirements. Each record is described in detail in Drawings numbered three through eight. Their respective lengths determine the size of the storage requirements of any data processing equipment employed. Chart No. 9 is included to show how the chaining sequence of the records is interfaced. This interaction of the records achieve for us two major functions, first to provide access to the facts and secondly, through this chaining sequence a single data base is developed to allow for implementation of the single information flow showing how a change, such as an engineering change, updates the whole data base, which is common to all. Thus, we accomplish one of our main objectives; one set of books is maintained.

5.5 BENEFITS OF THE PROPOSED SYSTEM

a. A Plan for Growth

A plan can be developed to begin implementation for each of the application subsystems leading to the single information flow system. The system can grow as Oceano-
### File Layout

#### Record and Work Areas

<table>
<thead>
<tr>
<th>RECORD NAME</th>
<th>ITEM MASTER RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIELD</td>
<td>VALUE</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Order Policy

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>MIN/MAX</th>
<th>MULTIPLIER</th>
<th>FIRST MATCH</th>
<th>SECOND MATCH</th>
<th>TWEED</th>
<th>SAFETY FACTOR</th>
<th>AVG/Demand</th>
<th>AVG/Forecast</th>
<th>IDNUM</th>
<th>GENCODE</th>
<th>SUM</th>
<th>SERIAL</th>
<th>PAR</th>
</tr>
</thead>
</table>

#### Lead Time

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>PRODUCTION</th>
<th>FIXED</th>
<th>SET UP</th>
<th>RUN</th>
<th>DATA</th>
</tr>
</thead>
</table>

#### Unit Cost

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>STANDARD COST</th>
<th>ACTUAL COST</th>
<th>LIST</th>
<th>NET</th>
</tr>
</thead>
</table>

#### Parts Usage History

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>TOTAL COMP</th>
<th>IPS</th>
<th>DGT</th>
<th>MFR</th>
</tr>
</thead>
</table>

#### Inventory on Hand

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>INVENTORY</th>
<th>TOTAL QTY</th>
<th>LOCATION</th>
</tr>
</thead>
</table>

#### Physical Inventory

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
<th>SERIAL</th>
<th>PAR</th>
</tr>
</thead>
</table>

---

**Figure SWAN 12**
OTHER FILES TO BE DESIGNED

PRODUCT STRUCTURE FILE
Purchased Requirements
Open Purchase Requisitions
Open Purchase Orders
Vendor Master
Purchase Master
Open Job Order Summary
Standard Routing
Work Center Master
Open Job Operation Detail
Tool Master

CHAINING CONCEPT

ITEM MASTER

<table>
<thead>
<tr>
<th>ITEM NUMBER</th>
<th>ADDRESS OF ROUTING</th>
<th>QUANTITY ON ORDER</th>
<th>FIRST ORDER NUMBER</th>
</tr>
</thead>
</table>

STANDARD ROUTING FILE

<table>
<thead>
<tr>
<th>ITEM NUMBER</th>
<th>OPERATION DESCRIPTION</th>
<th>WORK CENTER CHAIN</th>
</tr>
</thead>
</table>

SHOP ORDER 

| ORDER 

<table>
<thead>
<tr>
<th>ITEM NO.</th>
<th>NEXT ORDER NUMBER</th>
<th>CURRENT WORK CENTER</th>
</tr>
</thead>
</table>

WORK CENTER MASTER

<table>
<thead>
<tr>
<th>1st OPER. CENTER NUMBER</th>
<th>2nd OPER. CENTER NUMBER</th>
</tr>
</thead>
</table>

OPEN JOB OPERATIONAL DETAIL

<table>
<thead>
<tr>
<th>WORK CENTER NUMBER</th>
<th>ORDER NUMBER</th>
<th>OPERATION FOR THIS ORDER</th>
</tr>
</thead>
</table>

Figure SWAN 13
Procedure Flow Chart


Figure SWAN 14
Graphic grows. And Oceano-Graphic will obtain tangible results long before the total system is installed.

b. Standardization

Information is common to all—it is known and called by one description. One set of records makes this possible. Thus, it makes planning easier with standardized consistency in estimating and pricing.

c. Extensive Data Base

The record base has two important features: accessibility and accuracy. Information is accessible through inquiry to multiple points; detail is available through chaining to all related records. No longer will it be necessary to spend hours or days searching file drawers or ledger cards. Also, information is more accurate; it is updated in only one place. Standard transactions processed within each subsystem assure complete record maintenance.

d. Modular Program Design

Chart No. 10 shows the modular programming concepts. We can obtain tangible results before the total system is installed. This has been the major drawback of the single information flow concept. This will also allow a systematic plan of implementation with easily defined milestones for time and budget control.

e. All production information is now directed into a single channel. Levels of operating and management personnel are made more aware. With this assurance, the following should happen:
(1) Cost can be closely controlled, better surveillance over overtime hours, inventory, and machines—the key to cost reduction.
(2) More efficient planning.
(3) More time available to react to changes.
(4) Less waste, reduced information costs, more profits.

6.0 COST OF IMPLEMENTATION

The cost of implementation is a function of two variables. Machine costs of equipment capable of the level of mechanization required for this system represent one variable. Software cost, the cost of programming, represents a one-time cost necessary to the operation of the system.

6.1 HARDWARE COSTS

The hardware costs have not, for the purposes of this report, been precisely defined by the soliciting of quotes or bid proposals. However, with our experience directly in this field, a realistic estimate can be submitted. For instance, an IBM System 000/000 with a configuration to handle this application would cost $0,000 per month through the first years of operation. The addition of terminals as the need arises will increase the hardware costs to approximately $0,000 per month at later stages of development.

Operating and programming costs after the system is fully in operation will be approximately $000 per month. It is imagined this capability would be developed from within the current organization and no additional people added to the payroll.
6.2 SOFTWARE COSTS

The normal rule of thumb for predicting software cost for a new system such as this is to take the first year's cost for the hardware; in this case, $00,000. However, the sophistication of this system, which is considerably more elaborate than a payroll system, etc. (which is usually the makeup of software costs), can be accomplished for considerably less investment, since developed, canned programs, programs already written, can be used to minimize this expenditure. However, the canned programs still must be modified and patched to fit the Oceano-Graphic operation. We, therefore, estimate the cost of programming, debugging, and implementation of the system to be $00,000, approximately $0,000 per month for a period of ten months (Reference Drawing No. 11, Implementation Plan).
ACTIVITIES

Question 1. The existing system was evaluated from what points of view?

Answer: A. What are the objectives or premise for operation?
B. How was the premise put into operation, i.e., how did the planning take place?
C. In what manner was the plan implemented?
D. What was the measurement for determining how well the plan was being accomplished?
E. What was the medium for feedback and how did it affect the process?

Question 2. What was the object of the engineering data control?

Answer: To organize and maintain basic records.

Question 3. What are the basic records?

Answer: Item Masters, Product Structure, Standard Routing and Work Center Master.

Question 4. What other record is included under Engineering Data Control?

Answer: Open Job Order and Order Summary.

Question 5. What is the major objective of the Inventory Control System?

Answer: Record maintenance and updating of the inventory (it indicates when to order and how to order).
Question 6. What else is of importance as far as inventory control is concerned?

Answer: On-order fields are used in the Item Master so that Stock Status reports can be generated.

Question 7. What functions do the Sales Forecasting and Requirements Planning Connection perform?

Answer: The analysis of historical demand data, which may be stored on the item master to provide requirements planning with a gross finished product plan.

Question 8. What is the result of the merger of Requirements Planning with Inventory Control?

Answer: This makes it possible to determine the net time requirements, projected into time periods and schedule due dates.

Question 9. What other records are used?

Answer: Product Structure records are used to allow the breakdown of finished product items into individual components.

Question 10. What are the names of the four links that the Planned Orders are distributed to?

Answer: Purchasing, Electronic Assembly, Fabrication and Production Test.

Question 11. What are the four files involved when the Planned Orders reach the purchasing link?
Answer: Item Master, Purchase Master, Vendor Master and Open Purchase Order.

Question 12. How may a vendor be selected?
Answer: Through the use of a Purchase Master and a Vendor Master record.

Question 13. What is the purpose of the Capacity Planning Subsystem?
Answer: It identifies overloads far enough in the future to allow manpower planning and facility planning.

Question 14. What records are used to calculate the Order Start Date?
Answer: Standard Routing Records.

Question 15. Give the name of one of the key documents developed in this area according to the Production Model.
Answer: Work Center Load Report, projected by time period.

Question 16. What is the purpose of the Operation Scheduling Section?
Answer: It accepts orders which have gone through a releasing cycle from Capacity Planning, and schedules the work center within its short range time span.

Question 17. Shop Floor Control is the final subsystem in the flow. Give two of its functions.
Answer: A. It prepares the shop packets and other factory documentation.
B. It constructs the Open Job Summary and Operating
Detain Records so that the progress of the work can be reported.

Question 18. If we examine the set of Standardized Layout Records, we note that certain fields were used. Why?
Answer: The fields were used to allow Oceano-Graphic to utilize and control their production output requirements.

Question 19. What can the lengths of the records help determine?
Answer: The lengths can determine the size of storage requirements of any data processing equipment used.

Question 20. Chart 9 shows how chaining sequence of the record is interfaced. What two functions does this achieve?
Answer: A. This allows the development of the necessary retrieval functions to provide access to the facts.
   B. The chaining (or linking) allows the development of a single data base which means a single information concept.
(Make a check mark for your answer to Data Management Topics—included, not included, or irrelevant in the Swanson Study.)

FEASIBILITY STUDY

1. Basic Steps in Development of Business Systems

1. Problem definition
2. Application Research
3. Scope of study
   a. Organizational boundaries
   b. Objective of study
   c. Resources available for study
4. Objectives
5. Target dates
6. Study phase responsibility
7. Education of several departments
8. Management role

2. Factors Included in Systems Report to Management

1. Introduction
2. Description of study
3. Costs
4. Organizational changes

3. Systems and Procedures Covered

4. Functions of Systems and Procedures Department

1. Systems analysis and design
2. Forms design and control
3. Records management and retention
4. Report analysis
5. Preparation of written procedures
6. Work distribution
7. Process flow chart
8. Procedure flow chart
9. Work measurement
10. Time standards
11. Time and motion study
12. Forms control
   a. Retain only necessary forms
   b. Renew and reuse
   c. Proper design and manufacture
   d. One person assigned to forms control

5. Forms Design Basic Consideration

1. Purpose
2. Decision for specific information
3. Use and purpose
4. Logical placement of items
5. Physical placement of items
6. Analyzation of number of copies

6. Records Management
1. Creation and use of records
2. Distribution
3. Simplification of paperwork
4. Data retained
5. Filing procedures
6. Security

7. Company Manuals
1. Organizational
2. Policy
3. Operation
4. General information

8. Installation of Electronic Data Processing
1. Systems design
2. Personnel selection and training
3. File conversion
4. Forms design
5. Programming
6. Testing of programs
7. Facilities design and preparation
8. Scheduling and testing the system
9. Changeover
10. Feedback evaluation

9. Elements of Good Reports
1. Accurate
2. Clear
3. Relevant
4. Current

10. File Design
1. Volume consideration
2. Timing
3. Sorting

11. Fact Gathering Techniques of a System Study
1. Interview
2. Questionnaire
3. Review Records
12. Type of Information Gathered

1. Historical
2. Cost
3. Current procedure
4. Effectiveness
5. Relationship between departments
CONSTRUCTION

The Swanson Study uses a PERT type plan with Chart 11. The construction program can be used with this type of plan, and the cost program, which follows can be used to account for costs that occur daily.

Use the tape cassette to analyze this program. It will develop an introduction to the Basic Assembly Language used in this book.

The object of this program is to find the shortest time in which a construction project can be completed and to find the activities that limit the time for completion.

First a network for the project, such as Figure C-1 is drawn. Each arrow represents an activity. The numbered circles represent events. The numbers alongside the arrows are the work time, in days, for the activity. The diagram is laid out such that all activities leading to an event must be completed before any activities leading away from that event can begin. The ending number must be higher than the beginning event number.

The data is read into the computer, one activity per IBM card. The first sixteen spaces are reserved for the activity description, the next three spaces for the starting event number, then three spaces for the end event number, finally three spaces for the work time.

The data is stored in core storage, first the sixteen bytes of description, then two bytes for start event (packed format is used for all number data in the storage tables), two bytes for end event number, two bytes for normal work time. In addition, space is reserved in memory for start time, end time, slack time and an index. Two bytes of core storage are reserved for each activity in the table.
The program proceeds:

1. Initialize
2. Print header
3. Read all data cards and store information in table
4. Print out table
5. Sort the table on basis of lowest starting event number
6. Locate activity with the lowest starting event number
7. Locate and place in table TTWO all activities with end event numbers that match the start event numbers of (6)
8. Sort table TTWO on the basis of ending times
9. Subtract ending time of each activity from the ending time of the largest activity in table TTWO
10. Enter difference in slack time position
11. Enter the longest event time from table TTWO into start time for activity (6)
12. Compute end time for activity (6)
13. Index activity (6)
14. Transfer table TTWO and activity (6) back into main table
15. Test for last event
16. NO-back to (6)
17. Print out main table for second time
18. Sort main table on basis of lowest slack time first
19. Cut table off after a last event with no slack time
20. Sort new short table on basis of start event number, highest first
21. Print headers
22. Print out first activity in table (this activity is actually the last activity to be accomplished on the project)
23. Index first activity with a %%

24. Locate and print all activities with end event numbers the same start event as 21

25. Index each activity printed with a cret

26. Return to beginning of table, find first activity not indexed with %%, but indexed with a cret

27. Locate and print all activities with end event numbers the same as start event number 25

28. At last event end program
Construction—Flow Chart

Figure FC-02
Construction—Flow Chart

Figure FC-03
Construction—Flow Chart

Figure FC-01
Construction Program—Network Layout

Figure NET-01
Construction Program

Sort Routine

Figure SRT-01
## Construction Program Print Out

| ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 |
|------|----|------|---|------|----|------|---|------|----|------|---|------|----|------|---|------|----|------|
| ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 |
| ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 |
| ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 |
| ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 |
| ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 | ALVA | 10 | 0064 | 01 |

---

**Figure CON-01**
Figure CON-02
Construction Program

<table>
<thead>
<tr>
<th>SUM</th>
<th>RC</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM</td>
<td>9.1</td>
<td>FFHD</td>
</tr>
<tr>
<td>TOTI</td>
<td>SAP</td>
<td>1.1S.12,1.2.9</td>
</tr>
<tr>
<td></td>
<td>SAP</td>
<td>B1H=1121.T12H</td>
</tr>
</tbody>
</table>

**Figure CON-03**
Construction Program

Figure CON-04
Construction Program

Figure CON-05
This part of the program does the initialization and reads all the cards. Data is placed into a table in core storage by means of indexing a table. The index register used here is register 8. When all cards are read, a star (*) is placed at the end of the table. The table is then printed out by means of the BAS to OUT after the last card is read.
Construction Program—Analysis

The table is sorted on the basis of the start event number.

FIND—A compare is made to determine if it is the end of the table. If not, the starting time is moved to BINB. The end event of the table is compared to the start event in BINB. If they are equal, the start event is moved to BINC and then BINC is moved to TTWO.

The registers are incremented and a compare is made for the star. If no star has been reached, a branch is made back to FINA.

When a star is found for the end of the table, indexed by register 8, a star is placed in TTWO which is indexed by register 9.
Construction Program--Analysis

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

| 2570 | 1986 |
| 4774 | 1899 |
| 4710 | 1386 |
| 4714 | 1386 |
| 4716 | 1386 |

Figure CONA-03

ADDA---The work time is added to each start time and then placed back into TTWO as the new end time.

SOR2---Sorts the secondary table TTWO on the basis of the end events.
**Construction Program—Analysis**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM 1000</td>
<td>1044</td>
<td>Subtract ending time of each activity from the ending time of the longest activity in TTWO. Enter the difference, which is the slack time, in the slack time position. The index register is then reset and the beginning of the table is again loaded into register 8.</td>
</tr>
<tr>
<td>HOPE 46 46</td>
<td>0111</td>
<td>A compare is made with BINB (the beginning time and the end time). If they match, we return it to the main table.</td>
</tr>
<tr>
<td>INST 90 90</td>
<td>0112</td>
<td>Transfer table TTWO back to the main table. If it is the end of the main table a zero is zapped into slack time here because</td>
</tr>
</tbody>
</table>

**Figure CONA-04**
there is no slack time in the last event. Skip to the next page and print out the headings and the table. This time the switch at the end of section CONE send us to OUT for the next sort and print.

Construction Program—Analysis

The table is sorted on the basis of the lowest slack time first.

COMR—A star is placed at the point where the zero slack time ends.
Construction Program—Analysis

SOR4--The new short table is sorted on the basis of the highest start event number first. This table will contain all the events with no slack time. This means we start at the back of the critical path drawing.

SKIT--Skip to a new page, print out headers, and space. The address of the beginning of the main table is loaded into register 8. The first activity printed out here is actually the last activity of the drawing.

Figure CONA-06
LOPD—Index the first activity with a percent sign and add 30 to the main table. Reset the index register and check for a percent sign. If so, jump to ADVA; if not, check for a cret. If it is a cret, jump back to LOPD. This process now replaces each cret with a percent sign. The percent tells us how far we have gone, and the cret tells us the next adjacent event along the path that has no slack time.
## Construction Program—Output

### Output Number 01

<table>
<thead>
<tr>
<th>Activity</th>
<th>Start</th>
<th>End</th>
<th>Work</th>
<th>Start</th>
<th>Total</th>
<th>Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean Up</td>
<td>26</td>
<td>30</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sign Contract</td>
<td>26</td>
<td>30</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>See Lawyer</td>
<td>26</td>
<td>31</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sell</td>
<td>26</td>
<td>31</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memo</td>
<td>26</td>
<td>32</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rent</td>
<td>26</td>
<td>32</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collect</td>
<td>26</td>
<td>32</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finish</td>
<td>26</td>
<td>33</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End</td>
<td>26</td>
<td>34</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Second Time</td>
<td>13</td>
<td>16</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elevate</td>
<td>13</td>
<td>16</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bridge</td>
<td>13</td>
<td>16</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimate</td>
<td>13</td>
<td>16</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relate</td>
<td>13</td>
<td>16</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compose</td>
<td>15</td>
<td>18</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Composite</td>
<td>15</td>
<td>18</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runway</td>
<td>15</td>
<td>19</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ridiculous</td>
<td>15</td>
<td>21</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>January</td>
<td>17</td>
<td>20</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monday</td>
<td>17</td>
<td>21</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>February</td>
<td>17</td>
<td>21</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>March</td>
<td>18</td>
<td>25</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>April</td>
<td>18</td>
<td>22</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>May</td>
<td>18</td>
<td>23</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>June</td>
<td>19</td>
<td>25</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>July</td>
<td>20</td>
<td>25</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wednesday</td>
<td>22</td>
<td>28</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>August</td>
<td>18</td>
<td>22</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>December</td>
<td>21</td>
<td>26</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>September</td>
<td>21</td>
<td>26</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>October</td>
<td>21</td>
<td>26</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>November</td>
<td>21</td>
<td>26</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Veil</td>
<td>13</td>
<td>14</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sunday</td>
<td>13</td>
<td>16</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Railway</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Confusion</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>June</td>
<td>4</td>
<td>9</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lousy</td>
<td>6</td>
<td>10</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Effort</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applesauce</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kluge</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>izebert</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tuesday</td>
<td>25</td>
<td>24</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td>14</td>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unique</td>
<td>12</td>
<td>14</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tantalize</td>
<td>12</td>
<td>14</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Omitted</td>
<td>10</td>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Witten</td>
<td>10</td>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Silly</td>
<td>11</td>
<td>14</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optimum</td>
<td>8</td>
<td>11</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supple</td>
<td>8</td>
<td>12</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quiet</td>
<td>9</td>
<td>12</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>6</td>
<td>11</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finally</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gogy</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardly</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imagine</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saturday</td>
<td>22</td>
<td>26</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nearly Done</td>
<td>24</td>
<td>26</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thursday</td>
<td>25</td>
<td>29</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Friday</td>
<td>23</td>
<td>29</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure CON-07
### Construction Program—Output

**Output Number 02**

#### Antelope Valley Construction Company Activities Schedule

<table>
<thead>
<tr>
<th>Activity</th>
<th>Start Event</th>
<th>End Event</th>
<th>Work Time</th>
<th>Start Time</th>
<th>Total Time</th>
<th>Slack Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAVEMENT</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMISSION</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APPLESAUCE</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DENSITY</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EFFluent</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FINISH</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAME</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNK</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HARDLY</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMAGINE</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KNOCK</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOUDY</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAYBE</td>
<td>7</td>
<td>11</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NATOET</td>
<td>7</td>
<td>11</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTIMUM</td>
<td>8</td>
<td>11</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POMPEY</td>
<td>9</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUIET</td>
<td>10</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEIZED</td>
<td>10</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SILENT</td>
<td>11</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DANCE</td>
<td>12</td>
<td>16</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEATHERS</td>
<td>12</td>
<td>17</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tantalize</td>
<td>12</td>
<td>17</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECOND TIME</td>
<td>13</td>
<td>17</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXCAVATE</td>
<td>13</td>
<td>17</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERITY</td>
<td>13</td>
<td>17</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IYHNE</td>
<td>14</td>
<td>19</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPOSITE</td>
<td>15</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KINMAY</td>
<td>15</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOLDICIOUS</td>
<td>15</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BATTLE</td>
<td>16</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPRESS</td>
<td>16</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JANUARY</td>
<td>17</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEBRUARY</td>
<td>17</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARCH</td>
<td>18</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APRIL</td>
<td>18</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAY</td>
<td>19</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNE</td>
<td>20</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUGUST</td>
<td>20</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONDAY</td>
<td>20</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JULY</td>
<td>20</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCTOBER</td>
<td>20</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEMBER</td>
<td>20</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JANUARY</td>
<td>21</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEBRUARY</td>
<td>21</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARCH</td>
<td>22</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APRIL</td>
<td>22</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAY</td>
<td>23</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNE</td>
<td>23</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUGUST</td>
<td>23</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONDAY</td>
<td>23</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JULY</td>
<td>23</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCTOBER</td>
<td>23</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEMBER</td>
<td>23</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JANUARY</td>
<td>24</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEBRUARY</td>
<td>24</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARCH</td>
<td>25</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APRIL</td>
<td>25</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAY</td>
<td>26</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNE</td>
<td>26</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUGUST</td>
<td>26</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONDAY</td>
<td>26</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JULY</td>
<td>26</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCTOBER</td>
<td>26</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEMBER</td>
<td>26</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JANUARY</td>
<td>27</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEBRUARY</td>
<td>27</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARCH</td>
<td>28</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APRIL</td>
<td>28</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAY</td>
<td>29</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNE</td>
<td>29</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUGUST</td>
<td>29</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONDAY</td>
<td>29</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JULY</td>
<td>29</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCTOBER</td>
<td>29</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEMBER</td>
<td>29</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JANUARY</td>
<td>30</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEBRUARY</td>
<td>30</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARCH</td>
<td>31</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APRIL</td>
<td>31</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAY</td>
<td>32</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNE</td>
<td>32</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUGUST</td>
<td>32</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONDAY</td>
<td>32</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JULY</td>
<td>32</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCTOBER</td>
<td>32</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEMBER</td>
<td>32</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JANUARY</td>
<td>33</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEBRUARY</td>
<td>33</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARCH</td>
<td>34</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APRIL</td>
<td>34</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAY</td>
<td>35</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUNE</td>
<td>35</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUGUST</td>
<td>35</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONDAY</td>
<td>35</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JULY</td>
<td>35</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCTOBER</td>
<td>35</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEMBER</td>
<td>35</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure CON-08**
## Construction Program—Output

Output Number 03

### Antelope Valley Construction Company Critical Path Activities

<table>
<thead>
<tr>
<th>Activity</th>
<th>Start Event</th>
<th>End Event</th>
<th>Work Start Time</th>
<th>Work End Time</th>
<th>Total Time</th>
<th>Slack Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>END</td>
<td>33</td>
<td>34</td>
<td>1</td>
<td>40</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>FINISH</td>
<td>32</td>
<td>33</td>
<td>3</td>
<td>37</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>PAY</td>
<td>30</td>
<td>32</td>
<td>3</td>
<td>35</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>SIGN CONTRACT</td>
<td>26</td>
<td>30</td>
<td>3</td>
<td>31</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>CLEAN UP</td>
<td>23</td>
<td>30</td>
<td>6</td>
<td>28</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>SUNDAY</td>
<td>26</td>
<td>26</td>
<td>1</td>
<td>30</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>NOVEMBER</td>
<td>21</td>
<td>26</td>
<td>5</td>
<td>25</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>ART</td>
<td>18</td>
<td>23</td>
<td>7</td>
<td>23</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>REVOLVERS</td>
<td>15</td>
<td>21</td>
<td>7</td>
<td>23</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>RELATE</td>
<td>16</td>
<td>18</td>
<td>7</td>
<td>16</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>SECOND TIMES</td>
<td>13</td>
<td>16</td>
<td>2</td>
<td>14</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>UNCLE</td>
<td>14</td>
<td>15</td>
<td>1</td>
<td>17</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>VELITY</td>
<td>13</td>
<td>14</td>
<td>3</td>
<td>16</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>SILLY</td>
<td>11</td>
<td>14</td>
<td>4</td>
<td>13</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>ROTTEN</td>
<td>10</td>
<td>13</td>
<td>3</td>
<td>11</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>ROTTEN</td>
<td>10</td>
<td>13</td>
<td>3</td>
<td>11</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>OMITTED</td>
<td>10</td>
<td>11</td>
<td>2</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>KILLING</td>
<td>5</td>
<td>10</td>
<td>6</td>
<td>5</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>KILLING</td>
<td>4</td>
<td>10</td>
<td>6</td>
<td>5</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>DENSITY</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>APPEARANCE</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Figure CON-09
Figure AVC-01
### AVCC Program

```plaintext
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0248</td>
<td>ADD F YK C</td>
</tr>
<tr>
<td>0249</td>
<td>D217 FAK FAK</td>
</tr>
<tr>
<td>024A</td>
<td>F27 FPC FPC2</td>
</tr>
<tr>
<td>024B</td>
<td>F34 FPA FPC3</td>
</tr>
<tr>
<td>024C</td>
<td>D218 FAK FAK2</td>
</tr>
<tr>
<td>024D</td>
<td>D217 FAK FAK3</td>
</tr>
<tr>
<td>024E</td>
<td>F27 FPC FPC4</td>
</tr>
<tr>
<td>024F</td>
<td>F34 FPA FPC5</td>
</tr>
<tr>
<td>0250</td>
<td>F27 FPC FPC6</td>
</tr>
<tr>
<td>0251</td>
<td>F34 FPA FPC7</td>
</tr>
<tr>
<td>0252</td>
<td>F27 FPC FPC8</td>
</tr>
<tr>
<td>0253</td>
<td>F34 FPA FPC9</td>
</tr>
<tr>
<td>0254</td>
<td>F27 FPC FPC10</td>
</tr>
</tbody>
</table>

**Figure AVC-02**
<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0344 0256</td>
<td>F75D F718</td>
<td></td>
</tr>
<tr>
<td>0342 02F4</td>
<td>F7F2 F75D</td>
<td></td>
</tr>
<tr>
<td>034A F444</td>
<td>F107 F75D</td>
<td></td>
</tr>
<tr>
<td>0356 47F0</td>
<td>0022</td>
<td></td>
</tr>
<tr>
<td>0352 D25F</td>
<td>F760 F7cC</td>
<td></td>
</tr>
<tr>
<td>035A D20F</td>
<td>F720 F7b4</td>
<td></td>
</tr>
<tr>
<td>0358 D211</td>
<td>F74F F587</td>
<td></td>
</tr>
<tr>
<td>0356 0440</td>
<td>F33C</td>
<td></td>
</tr>
<tr>
<td>0356 D277</td>
<td>F60A F403</td>
<td></td>
</tr>
<tr>
<td>035C F376</td>
<td>F781 F753</td>
<td></td>
</tr>
<tr>
<td>035A D20C</td>
<td>F620 F7C8</td>
<td></td>
</tr>
<tr>
<td>0358 D211</td>
<td>F618 F47C</td>
<td></td>
</tr>
<tr>
<td>035C D277</td>
<td>F43F F6d4</td>
<td></td>
</tr>
<tr>
<td>0356 0400</td>
<td>F33C</td>
<td></td>
</tr>
<tr>
<td>0356 D277</td>
<td>F60A F403</td>
<td></td>
</tr>
<tr>
<td>0414 F444</td>
<td>PTC2 F753</td>
<td></td>
</tr>
<tr>
<td>0412 F644</td>
<td>FTC2 F726</td>
<td></td>
</tr>
<tr>
<td>0420 D016</td>
<td>F40D F709</td>
<td></td>
</tr>
<tr>
<td>0426 D040</td>
<td>F7C2 F7C2</td>
<td></td>
</tr>
<tr>
<td>042C D211</td>
<td>F618 F404</td>
<td></td>
</tr>
<tr>
<td>0432 D277</td>
<td>F43F F6d4</td>
<td></td>
</tr>
<tr>
<td>0434 0400</td>
<td>F33C</td>
<td></td>
</tr>
<tr>
<td>0436 D277</td>
<td>F404 F604</td>
<td></td>
</tr>
<tr>
<td>0442 F376</td>
<td>F796 F756</td>
<td></td>
</tr>
<tr>
<td>0446 D20C</td>
<td>F633 F7c4</td>
<td></td>
</tr>
<tr>
<td>0446 0400</td>
<td>F33C</td>
<td></td>
</tr>
<tr>
<td>0454 D211</td>
<td>F618 F682</td>
<td></td>
</tr>
<tr>
<td>0458 D277</td>
<td>F43F F604</td>
<td></td>
</tr>
<tr>
<td>0460 0400</td>
<td>F33C</td>
<td></td>
</tr>
<tr>
<td>0466 D277</td>
<td>F606 F603</td>
<td></td>
</tr>
<tr>
<td>0464 F374</td>
<td>F7a4 F7a7</td>
<td></td>
</tr>
<tr>
<td>0470 D20C</td>
<td>F649 F7c4</td>
<td></td>
</tr>
<tr>
<td>0476 D20C</td>
<td>F449 F7a7</td>
<td></td>
</tr>
<tr>
<td>047C D211</td>
<td>F618 F4C4</td>
<td></td>
</tr>
<tr>
<td>0482 D277</td>
<td>F42C F404</td>
<td></td>
</tr>
<tr>
<td>0484 0400</td>
<td>F33C</td>
<td></td>
</tr>
<tr>
<td>0492 D277</td>
<td>F404 F603</td>
<td></td>
</tr>
<tr>
<td>0494 0404</td>
<td>F33c</td>
<td></td>
</tr>
<tr>
<td>049C D277</td>
<td>F606 F603</td>
<td></td>
</tr>
<tr>
<td>0492 F644</td>
<td>PTC2 F767</td>
<td></td>
</tr>
<tr>
<td>0498 F644</td>
<td>PTC2 F767</td>
<td></td>
</tr>
<tr>
<td>049E D20E</td>
<td>F570 F759</td>
<td></td>
</tr>
<tr>
<td>04A4 D200</td>
<td>F763 F7c4</td>
<td></td>
</tr>
<tr>
<td>04B0 D211</td>
<td>F618 F404</td>
<td></td>
</tr>
<tr>
<td>04B6 D277</td>
<td>F43F F606</td>
<td></td>
</tr>
<tr>
<td>04BE 0408 F33C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure AVC-03**
Figure AVC-04
### Antelope Valley Construction Company
#### Cash Requirements Prediction

**Week Number 01**
- **Income**: $3,000.00
- **Expense**: $5,000.00
- **Difference**: $2,000.00
- **Income to Date**: $3,000.00
- **Expense to Date**: $5,000.00
- **Difference to Date**: $2,000.00

**Week Number 02**
- **Income**: $4,500.00
- **Expense**: $13,000.00
- **Difference**: $8,500.00
- **Income to Date**: $9,500.00
- **Expense to Date**: $20,500.00
- **Difference to Date**: $10,500.00

**Week Number 03**
- **Income**: $12,000.00
- **Expense**: $3,500.00
- **Difference**: $8,500.00
- **Income to Date**: $21,500.00
- **Expense to Date**: $23,500.00
- **Difference to Date**: $2,000.00

**Week 04**
- **Income**: $4,955.67
- **Expense**: $6,750.00
- **Difference**: $1,794.33
- **Income to Date**: $30,445.67
- **Expense to Date**: $30,250.00
- **Difference to Date**: $195.67

**End AVCC Master**

---

Figure AVC-06
SECTION II

DATA STRUCTURES
II DATA STRUCTURES

The Swanson Study was presented to help the reader become aware of a need for other programming tools other than the higher level languages. If he is going to be able to function at the level of this study, it will be well to master certain aspects of Data Structures.

The object of this section of the book is to provide the material so that the reader can develop his own information system, or at least work at the level of proficiency the Swanson Study demands.

This introductory section presents a definition of data structures by depicting the recommendations of the Committee of the Association of Computing Machinery for a college level course in data structures. This allows the reader to choose other subjects he might desire and pursue them by using the bibliography in the appendix.

HISTORY

Prior to the 1950's programmers were concerned with both memory allocation and the algorithm. Each programmer accepted the fact that he must exercise adequate control over fast-access storage, if the program were too large for the storage and he had to use auxiliary.

During the 1950's higher level languages were introduced, and the programmer became more involved with problem solving than with the machine and its various aspects. The programs grew in size and so did the problem of storage control and storage allocation.

Due to the shielding effect of the higher level languages, the programmer moved further and further from the reality of the efficiency
and inefficiency of the machine, and the requirement was for larger and more expensive memories. When the cost became so great, the realization came that it was time for the programmer to face reality and go back and work with such elements as information structures and storage allocation.

It would appear that we have come back to a point that has characteristics similar to the pre-1950's.

It will now be necessary for the programmer to put forth the extra effort to master the areas he avoided even though they may appear to be more difficult in some instances.

The term "INFORMATION STRUCTURE" as used in this book relates to the contents of the course of study presented by the Association of Computing Machinery in 1968. This term is used to infer the structural relationships between various data elements.

The Curriculum Committee on Computer Science of the Association for Computing Machinery has offered recommendations for a college level course, entitled "Data Structures".

1. Basic Concepts of Data
   a. Representation of information as data inside and outside the computer, bits, nodes and data elements
   b. Data files and tables
   c. Names, values, environments
   d. Use of pointer or linkage variables to represent data structures

e. Identifying entities about which data is to be maintained
f. Selecting data nodes and structures which are to be used in problem solution
g. Storage media, storage structures, encoding data and transformations from one medium and/or code to another
h. Alternative representations of information and data
i. Packing, unpacking and compressing data
j. Data formats, data description languages, specifications of data transformations

2. Linear Lists and Strings
   a. Stacks, last-in first-out, first-in first-out, double ended, and other linear lists
   b. Sequential versus linked storage
c. Single versus double link circular lists
d. Character strings of variable length
e. Word packing, part word addressing
f. Pointer manipulation
g. Insertion, deletion and accessing of list elements

3. Arrays and Orthogonal Lists
   a. Storage of rectangular arrays in a one dimensioned media
   b. Storage mapping functions
c. Direct and indirect address computation
d. Space requirements
e. Set up time
f. Accessing time and dynamic relocation times
g. Storage and accessing triangular arrays
h. Tetrahedral arrays
i. Space matrices

4. Tree Structures
   a. Trees, subtrees, ordered trees, free trees, oriented trees, binary trees
   b. Representation of trees using binary trees
   c. Sequential techniques
   d. Threaded lists
   e. Insertion, deletion and accessing of elements of trees
   f. Relative referencing, finding successors and predecessors
   g. Walking through trees
   h. Examples of tree structures such as algebraic formulas, arrays and other hierarchic data structures (PLI/and COBOL)

5. Storage Systems and Structures
   a. Behavioral properties of Unit record (card)
   b. Random access (core)
   c. Linear (tape)
   d. Intermediate (disk and drum)
   e. Storage media including cost, size, speed, reusability
   f. Inherent record and file structure
   g. Deficiencies and interrelation of these properties
   h. Influence and machine structure--addressing on data structuring
   i. Hierarchies of storage, virtual memory, segmentation, paging and bucketing
   j. Influence of data structures and data manipulation on
storage systems

k. Associative structures, both hardware and software

6. Storage Allocation and Collection

7. Multi-linked Structures

8. Sorting (ordering techniques)
   a. Radix sorting
   b. Radix exchange sorting
   c. Merge sorting
   d. Bubble sorting
   e. Address table sorting
   f. Topological sorting
   g. Comparative efficiency of sorting techniques
   h. Effect of data structures and storage structures on sorting techniques

9. Symbol Table and Searching

10. Data Structures in Programming Languages
    a. Compile time and run time data structures needed to implement source language data structures of programming languages
    b. Linkage between partially executed procedures
    c. Data structures of co-routines, scheduled procedures, and control structures
    d. Storage management of data structures in procedure oriented languages
    e. Examples of higher level languages which include list processing and other data structure features

11. Formal Specification of Data Structures
a. Specification of syntax of classes of data structures
b. Predicate selectors and constructors for data manipulation
c. Data definition facilities
d. Programs as data structures
e. Computers as data structures and transformations
f. Formal specification of semantics
g. Formal systems viewed as data structures

12. Generalized Data Management Systems

a. Structures of generalized data management systems
b. Directory maintenance
c. User languages (query)
d. Data description maintenance
e. Job management
f. Embedding data structures in generalized data management systems
g. Examples of generalized data management systems and comparison of system features

II DATA STRUCTURES

If the reader will notice the numbering sequence used for this section, it does contain a certain structuring. The section begins with Linear Lists and ends with Virtual Memory.

There was a definite reason in presenting the subjects in this particular order, and that was one of logical development of the tools needed to create information systems.
The basic information starts with lists (and links), which will partially answer the problem of linking, created by the Swanson Study. If the reader will examine the definition of data structures, he will see that Files are listed at lb. Files are listed in this section at 2.4 because they fit logically into the development of the material for an information system.

The final culmination of this section ends with Dynamic Storage Allocation and Virtual Memory. They are presented as a little icing for the cake, since certain programming positions do not require an awareness of them, however they are presented for the reader who wants to do just a little bit more.

II DATA STRUCTURES

2.1 Introduction

2.2 Linear Lists
   2.2.1 Stacks, Deques and Queues
   2.2.2 Linked Lists

2.3 Trees
   2.3.1 Hierarchical Ring Structure
   2.3.2 Traversing Trees - Preorder and Post Order
   2.3.3 Directories
      2.3.3.1 Structured Tree Directory
      2.3.3.2 Random Ordered Directories

2.4 Files
   2.4.1 Sequential Files
   2.4.2 Inverted Files
      2.4.2.1 Creation, Maintenance and Deletion of Keys
2.4.2.2 Logical Operators
2.4.2.2.1 Logical Operators
2.4.2.2.2 Logical Operator "OR"
2.4.2.2.3 Logical Operator "AND"

2.4.3 Multilist File
2.4.3.1 Deletion of Records and Addition of Keys

2.4.4 Cellular Multilist File

2.5 Sorting

2.5.1 Sorting Methods
2.5.1.1 Key Sort
2.5.1.2 Detached Key Sort
2.5.1.3 Non Detached Key Sort

2.5.2 Radix Sort
2.5.3 Quicksort
2.5.4 Topological Sort
2.5.5 Quadratic Sort
2.5.6 Calculated Address Sort
2.5.7 Sorting by Insertion
2.5.8 Interchange Sort

2.6 Merging

2.6.1 Two Way Merge
2.6.1.1 Merging Ordered Files with Subfiles
2.6.1.1.1 Merging Step Downs

2.7 Search

2.7.1 Linear Search
2.7.2 Binary Search

2.8 Dynamic Storage Allocation
List Formats

An important aspect of the list languages is that the original list of available space must be either set up automatically by the system or manually by the programmer. This original space amounts to a pool or block of storage which the program can draw from as needed.

Three kinds of fixed format lists have been adopted by various writers. The format presented here was originally named by Knuth:

1. QUEUES enable opposite end accessibility; input one end and output the other end.
2. DEQUES enable double end accessibility; a word or record can be added or deleted at either end.
3. STACKS enable double end accessibility; a word or record can be added to one end.

It is important to note that with the formats presented here, it is not possible to access records that are located in the center of either method.

The Que has two ends, and the records are entered at the top.
and withdrawn from the bottom. This method is generally used when records are to be serviced in the order of receipt, (FIFO, which is first-in first-out). This method also is used in accounting. A Queue is used for a list of areas to receive records from sequential files, and it is also used as a list of areas to be written on a sequential file or files.

DEQUE

It is possible to insert or delete a record from either the top or the bottom of the deque. This could be called last-in first-out or first-in first-out.

STACK

A stack takes out the most recent item that was put in. It uses the last-in first-out method. This method is one that is used more for list than the other methods. The list of available space is
one application for this method.

List of Available Space

A linked list indicates some type of instrument to control the space not in use. As evidenced by the language SLIP, written by Wizenbaum, a List of Available Space is quite necessary. It is essentially a stack.

![Diagram of a stack with IN and OUT arrows pointing to the top and bottom respectively.]

The last (top) record is the first one out. All nodes that are not being used are linked by this list.

Linked Lists

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>ITEM</th>
<th>LINK</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ITEM 1</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>ITEM 2</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>ITEM 3</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>ITEM 4</td>
<td>*</td>
</tr>
</tbody>
</table>
Go Through a Stack

The linked list is much more flexible than the sequential list. The addresses 10, 11, 12 and 13 are addresses in memory, and the figure (*) is a null link, which stops the action.

The program requires a link variable that points to the address number 10, and from address 10, all other items on the list are easily found.

```
*  G O  T H R O U G H  A  S T A C K
*  REGISTER 9 POINTS TO TOP OF STACK
*  REGISTER 10 HAS RETURN ADDRESS FROM SUBR.
*  REGISTER 11 HAS RETURN ADDRESS FROM
*  SUBROUTINE VIST

START

*  USING *,0

STAR

HAS  11, VIST  GO TO SUB FOR WORK TO BE DONE

*  9, H4

AM

9, 0(0,9)  LOAD REG 9 TO POINT TO NEXT ITEM IN STACK

*  H0(TM

CH  9, H0(TM

IS IT BOTTOM OF STACK =

2,10

YES, RETURN FROM SUBR.

15, STAR

END  STAR
```
Refer to the Swanson Study to Chart #9, which depicts the Bill of Material retrieval. Also refer to File Page #4, which links the various files together. The above drawing is intended to carry the ideas the two Swanson charts contain, and prepare the reader for the linking process.

Activities

1. Duplicate the drawing of the linkage of File Page #4.
2. Duplicate the drawing of Chart #9.
3. Listen to the tape cassettes that pertain to this section, and review the Basic Assembler set of instructions.
ADD TO TOP OF A STACK

REGISTER 8 POINTS TO AVAILABLE STORAGE
REGISTER 9 POINTS TO TOP OF STACK
REGISTER 10 HAS RETURN ADDRESS FROM SUBR.

A
START 0
USING *,0

STAR CH 8, LIMIT
         ANY MORE AVAILABLE SPACE
HC 2, OVFL
     NO, HRANCH TO OVERFLOW
MVC 0(4, 8), DATA STROKE DATA IN AVAIL.
STH 8, TEMP
     SAVE POINTER FOR NEW TOP
     OF STACK
AH 8, H4
     REG. 8 POINTS TO LINK OF
     AVAIL.
STH 9, 0(0, 8)
     AVAIL LINKED TO TOP OF
     STACK
LM 9, TEMP
     REG. 9 POINTS TO LINK OF
     AVAIL
AH 8, H2
     REG. 8 POINTS TO NEW
     AVAILABLE STORAGE
BGR 15, 10
     RETURN FROM SUBROUTINE
END STAR

Figure I-4
The disadvantage of the extra space used by the links may be circumvented by including several items to each link, and a link need not take a large amount of space. It would take only one byte on a byte oriented computer. It is also possible to create an overlap of tables and share common parts. (This will be discussed later.)

This example of a one way link assumes that in any type of search, the direction can proceed forward only.

Delete From Top of Stack

```
* REG. 9 POINTS TO LINK OF ITEM A
* REG. 10 HAS RETURN ADDRESS FROM SUBROUTINE
A
  START 0
  USING #0
STAR
  AH   9,4H4   POINT TO LINK OF A
  LM   9,0(0,9) REGISTER 9 POINTS TO B
  ( NEW TOP OF STACK )
  CH   9,HITM  BOTTOM OF STACK =
  BC   2,UNFL  YES, GO TO UNDERFLOW
  BCR  15,10   RETURN FROM SUBROUTINE
END       STAR
```

Figure L-5

Delete From Inside a Stack

The deletion of an item (or items) is easy, since the deletion requires a change in the link to address 13. The same change with a sequential file would entail a deletion of the item in address 12, and a move up of the following items.
The insertion of an item into a list is easy because the inserted node can reside at any available location.
Delete From Left End of List

Delete From Left End of List

REG. 8 POINTS TO AVAILABLE STORAGE
REG. 9 POINTS TO LEFT LINK OF ITEM C
REG. 10 HAS RETURN ADDRESS FROM MAIN SUBR.

START 0
USING *, 0

CH 8, LIMIT ANY MORE AVAILABLE STORAGE
HC 2, OVL $, G0 TO OVERFLOW
MVC 0(2), 0(9) STORE RIGHT LINK TO B
     INTO LEFT LINK OF AVAIL
AH 8, H2 POINT TO DATA OF AVAIL
MVC 0(2), 0(9) STORE DATA IN AVAIL
AH 8, H2 POINT TO RIGHT LINK OF AVAIL
STH 9, O(0, 8) STORE LINK TO C INTO
     RIGHT LINK OF AVAIL
AH 8, H2 POINT TO NEW AVAILABLE
     STORAGE
MVC 0(2), 0(8) STORE LINK TO AVAIL
     INTO LEFT LINK OF C
SH 8, H6 POINT TO LEFT LINK OF AVAIL
LH 9, O(0, 8) REG. 9 WILL POINT TO
     RIGHT LINK OF B
STH 8, O(0, 9) RIGHT LINK OF B WILL
     POINT TO LEFT LINK OF AVAIL
AH 8, H4 POINT TO RIGHT LINK OF AVAIL
LH 9, O(0, 8) REG. 9 WILL POINT TO
     LEFT LINK OF C
AH 8, H2 POINT TO NEW AVAILABLE ST.
BCR 15, 10 RETURN FROM SUBROUTINE

END STAR

Delete From Left End of List

10 11 12 13
ITEM 1 ITEM 2 ITEM 3 ITEM 4

arrow
Go Through a Doubly Linked List

Reg 9---points to left end of the list

Reg 10---contains the return address from the main subroutine

Reg 11---contains the return address from the subroutine Visit.

(This subroutine performs some function and then returns us to the main program or subroutine we were in before the jump to Visit.)
Add to Left End of Doubly Linked List

Reg 8—points to available storage
Reg 9—points to left end of list
Reg 10—contains the return address from the subroutine
Doubly Linked Lists

Reg 8—points to left link of available storage

Reg 9—points to left link of an item in list

Reg 10—contains the return address from the subroutine

Insert Into List Between B and C (when Reg 9 points to left link of C)
* INSERT INTO A LIST BETWEEN ITEMS B AND C
* REG. 8 POINTS TO AVAILABLE STORAGE
* REG. 9 POINTS TO LEFT LINK OF ITEM C
* REG. 10 HAS RETURN ADDRESS FROM MAIN SUBR.

A
START 0
USING #,0

START
CH 8,LIMIT ANY MORE AVAILABLE STORAGE
MC 2,IVFL NO, GO TO OVERFLOW
MVC 0(2,8),0(9) STORE RIGHT LINK TO B INTO LEFT LINK OF AVAIL
AH 8,H2 POINT TO DATA OF AVAIL
MVC 0(2,8),DATA STORE DATA IN AVAIL
AH 8,H2 POINT TO RIGHT LINK OF AVAIL
STH 9,0(0,8) STORE LINK TO C INTO RIGHT LINK OF AVAIL
AH 8,H2 POINT TO NEW AVAILABLE STORAGE
MVC 0(2,9),0(8) STORE LINK TO AVAIL INTO LEFT LINK OF C
SH 8,H6 POINT TO LEFT LINK OF AVAIL
LH 9,0(0,8) REG. 9 WILL POINT TO RIGHT LINK OF B
STH 8,0(0,9) RIGHT LINK OF B WILL POINT TO LEFT LINK OF AVAIL
AH 8,H4 POINT TO RIGHT LINK OF AVAIL
LH 9,0(0,8) REG. 9 WILL POINT TO LEFT LINK OF C
AH 8,H2 POINT TO NEW AVAILABLE ST.
BCR 15,10 RETURN FROM SUBROUTINE
END STAR

Delete From Inside of List

![Diagram of list deletion](image)

REG 8 AVL
REG 9 A(0)
REG 10 RETURN
REG 11 TEMP
DELETE FROM INSIDE OF A LIST

REG. 9 POINTS TO THE LEFT END OF THE LIST
REG. 10 HAS RETURN ADDRESS FROM MAIN SUBR.

START 0
USING * . 0

STAR
STM 9, TEMP SAVE REG. 9
AH 9, H4 POINT TO RIGHT LINK OF A
LH 11, 0(0, 9) POINT TO LEFT LINK OF B
AH 11, H4 POINT TO RIGHT LINK OF B
MVC 0(2, 9), 0(11) RIGHT LINK OF A WILL
     POINT TO LEFT LINK OF C
LH 9, 0(0, 11) REG 9 WILL POINT TO THE
     LEFT LINK OF C
SH 11, H4 POINT TO LEFT LINK OF B
MVC 0(2, 9), 0(11) LEFT LINK OF C WILL
     POINT TO RIGHT LINK OF A
CH 9, REND IS IT THE RIGHT END OF
     THE LIST
BC 2, UNFL YES, GO TO UNDERFLOW
LH 9, TEMP RESTORE REG. 9 0 POINT
    TO LEFT LINK OF A
BCR 15, 10 RETURN FROM SUBROUTINE
END: STAR

Figure L-9

Activities

1. Flowchart example #1 "Go Through Stack"
2. Flowchart "Add To Top of Stack"
3. Flowchart "Delete From Top of Stack"
4. Flowchart "Insert Into Stack Between B and C"
5. Flowchart "Add to Left of Doubly Linked List"
6. Flowchart "Delete From Inside of List"
7. Flowchart "Doubly Linked Lists"
Trees

Lists and strings are understood easier, developing the hierarchal structure idea with trees.

Lists*** (((d))) (Q, R, ), L, () (PTA))

The comma and parentheses become quite important in depicting atoms from the list.

![Figure T-1]

The asterisks help to define the recursion. Here, the list contains the list (((d))) which consists of the list ((d)), which consists of list (d), which consists of the atom d. The asterisks indicate a blank or null.

This idea exists in the Dewey Decimal System for the libraries:

000-099 General Works
100-199 Philosophy, Psychology, Ethics
200-299 Religion and Mythology
300-399 Sociology (Economics, Civics, Education, Vocations)
400-499 Philology (Language, Dictionaries, Grammar)
500-599 Science
600-699 Useful Arts (Medicine, Engineering, Agriculture, Radio, Aviation, etc.)
900-999 History, Geography, Biography, Travel
Each of these ten main classes is broken up into more specialized fields. For example, class 600-699, Useful Arts, is broken into ten classifications including Medicine, Engineering, and Manufacturing. Each of these divisions is further subdivided.

This hierarchical tree idea exists in the paging scheme of this text. For example:

1.1
1.2
1.1.2

The idea also exists in set theory. For example, it exists in nested sets:

```
\( \{ (a, (b, (c)) \} \) \)
```

Figure T-2

A Combination of Sets and Trees

```
Figure T-3
```
The hierarchical idea allows us to combine sets and trees. For example:

Let $A$ — the set 123
Let $B$ — the set 242
Let $C$ — the set 345

With the help of a hierarchical tree, we can find $A \times B \times C$.

We see that $A \times B \times C$ consists of the ordered triplets to the right of the tree.

**Tree Hierarchical Structures**

![Figure T-4](image)

The idea of a tree is presented here to form a concept of structure that might exist within core memory.

![Figure T-5](image)
2.3.1 Hierarchical Ring Structure

The tree nodes are divided into three parts, which can be adjusted to fit most computers. This search procedure scans the descendants of the head (root) by following the chain of part two of each element (the pointer in part two). If a match is found on one level a branch can be made to the next level by means of the address of the pointer of part three of the element.

It is easy to insert a new node here because the chain can be broken and then the pointers can be changed to effect a new link.

This arrangement of the node, Figure H-1, amounts to the insertion of an atom in the first element of the node. The second part links to the brother on the same level, and the third part of the node links to the next structural level of descendancy.

The complete ring is made, which allows the return to the head of the list, and at the same time facilitates entering the ring at any point.
Algebraic Formulas

Algebraic formulas are better understood with the help of a tree hierarchy structure.

Figure H-2
Double Ring Link to Tree

This type of ring structure represents a double link type structure, which allows a much freer movement, but does take more space due to the pointers.

Figure H-3
Traversing Trees

Traverse a Tree in Preorder Sequence

Figure TT-1
Traverse the Tree in Preorder

A, B, C, D, E, F, G, H

Left Link = 2 bytes

Right Link = 2 bytes

Middle Link = 2 bytes

Data = 2 bytes

Null Link = *
Traverse a Tree in Preorder

* REG. 4 POINTS IN LEFT LINK OF ROOT OF TREE
* REG. 10 IS USED TO HANDLE DATA
* REG. 11 IS USED AS AN AUXILIARY REGISTER
* REG. 12 CONTAINS RETURN ADDRESS FROM SUBR.

A START

USING =, 0

STAR
SR 11, 11 CLEAR REG. 11
LH 10, 0(0, 9) LOAD REG. 10 FROM LEFT LINK OF ROOT

CH 10, NULL IS IT A NULL LINK =
MCR 8, 17 YES, RETURN FROM SUBR.
AH 9, H2 NO, POINT TO MIDDLE LINK
STH 12, 0(9) STORE RETURN ADDRESS IN MIDDLE LINK OF ROOT

SH 9, H2 POINT TO LEFT LINK
HAS 13, VIST GO TO SUBR. FOR WORK TO BE DONE
POS
AH 9, H6 POINT TO RIGHT LINK
LH 10, 0(0, 9) LOAD REG. 10 FROM R. LINK
CH 10, NULL IS IT NULL OR POSITIVE
MC 11, #+6 YES
SR 11, 10 NO, MAKE R.L. POSITIVE
STH 11, 0(9) STORE POS. NO. IN R.LINK
SR 11, 11 CLEAR REG. 11
SH 9, H6 POINT TO LEFT LINK
LH 9, 0(0, 9) POINT TO LEFT LINK OF NEXT NODE

NEXT BAS 13, VIST GO TO SUBR. FOR WORK TO BE DONE
LH 10, 0(0, 9) LOAD REG. 10 FROM L.LINK
CH 10, NULL IS IT A NULL LINK =
BC 7, POS NO, GO TO MAKE RIGHT LINK POSITIVE

NXTR
AH 9, H6 POINT TO RIGHT LINK
LH 10, 0(0, 9) LOAD REG. 10 FROM R.LINK
CH 10, NULL IS IT A NULL LINK =
MC 13, MIDL BRANCH IF REG. 10 NOT POS
SR 11, 10 NEGATE
STH 11, 0(9) STORE THE NEGATIVE NO. IN RIGHT LINK

LH 9, 0(0, 9) POINT TO NEXT NODE
AC 17, NEXT
MIDL
SH 9, H4 POINT TO MIDDLE LINK
LH 9, 0(0, 9) POINT TO NEXT NODE
AC 15, NXTR
END STAR

Figure TT-3
Traverse a Tree in Post Order

Figure TT-5
Post Order

Traverse the Tree in Post Order

D, C, B, F, E, G, A, H
Traverse a Tree in Post Order

- **REG. 0** points to left link of root of tree
- **REG. 10** is used to handle data
- **REG. 11** is used as an auxiliary register
- **REG. 12** contains return address from subr.

**START**

**STAR**

<table>
<thead>
<tr>
<th>Action</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>11,11</td>
</tr>
</tbody>
</table>
| LH     | 10,0(0,9)| Load reg. 10 from left
| CH     | 10, null | Is it a null link =
| RC     | 8, 12    | Yes, return from subr.
| AH     | 9, H2    | No, point to middle link
| STH    | 12, 0(0,9) | Store return address in middle link of root

**POS**

<table>
<thead>
<tr>
<th>Action</th>
<th>Contents</th>
</tr>
</thead>
</table>
| SH     | 9, H2    | Point to left link
| AH     | 4, H4    | Point to right link
| LH     | 10, 0(0,9)| Load reg. 10 from r. link
| CH     | 10, null | Is it NULL or POSITIVE
| HC     | 11, X+6  | YES
| SR     | 11, 10   | NO, make r. link POSITIVE
| STH    | 11, 0(0,9) | Store pos. no. in r. link
| SR     | 11, 11   | Clear reg. 11
| SH     | 9, H6    | Point to left link

**NEXT**

<table>
<thead>
<tr>
<th>Action</th>
<th>Contents</th>
</tr>
</thead>
</table>
| LH     | 9, 0(0,9) | Point to left link of next node
| LH     | 10, 0(0,9)| Load reg. 10 from l. link
| CH     | 10, null | Is it a null link =
| HC     | 7, POS   | No, go to make right link POSITIVE

**VISI**

<table>
<thead>
<tr>
<th>Action</th>
<th>Contents</th>
</tr>
</thead>
</table>
| MAS    | 13, VIST | YES, go to subr. for work to be done
| AH     | 9, H6    | Point to right link
| LH     | 10, 0(0,9)| Load reg. 10 from r. link
| CH     | 10, null | Is it NULL or NEGATIVE
| HC     | 2, NEG   | NO, go to set r. link NEG

**MIDL**

<table>
<thead>
<tr>
<th>Action</th>
<th>Contents</th>
</tr>
</thead>
</table>
| SH     | 9, H4    | Point to middle link
| LH     | 9, 0(0,9) | Point to next node
| AH     | 9, H6    | Point to right link
| LH     | 10, 0(0,9)| Load reg. 10 from next node
| CH     | 10, null | Is it NULL or NEGATIVE
| HC     | 4, MIDL  | YES, go to middle link
| SH     | 9, H6    | NO, point to left link
| HC     | 15, VISI | Go for work to be done

**NEG**

<table>
<thead>
<tr>
<th>Action</th>
<th>Contents</th>
</tr>
</thead>
</table>
| SH     | 11, 10   | NEGATE
| STH    | 11, 0(0,9) | Store neg. no. in r. link
| HC     | 15, NEXT | Go to next node

END

**STAR**

**Figure TT-8**
Add to a Tree Between B and C

Add to a Tree Between Nodes B and C

REG. 8 Points to Available Storage
REG. 9 Points to Node B

STAR

START 0
USING *

STAR

LH 10,0(0,9) SAVE LEFT LINK OF B
STH 8,0(0,9) LEFT LINK OF B WILL POINT TO AVAIL

STH 10,0(0,8) LEFT LINK OF AVAIL WILL POINT TO C

LH 9,0(0,8) REG. 9 POINTS TO LEFT LINK OF C

AH 9,H2 REG. 9 POINTS TO MIDDLE LINK OF C

LH 10,0(0,9) SAVE MIDDLE LINK OF C
STH 8,0(0,9) MIDDLE LINK OF C POINT TO LEFT LINK OF AVAIL

AH 8,H2 REG. 8 POINTS TO MIDDLE LINK OF AVAIL

STH 10,0(0,8) MIDDLE LINK OF AVAIL POINTS TO L.L. OF B

END STAR

Figure TT-9
Delete Node C From A Tree

**DELETE NODE C FROM A TREE**

**REG.9 POINTS TO LEFT LINK OF R**

**START**

```
START 0
USING *,0
```

```
STAR   LM  8,0(0,9)  REG.8 POINTS TO L.L. C
       MVC 0(2,9),0(8)  L.L. IF H WILL POINT
                      TO LEFT LINK OF D
       LM  9,0(0,8)  REG.9 POINTS TO L.L. D
       AH  8,92      REG.8 POINTS TO M.L. C
       AH  9,92      REG.9 POINTS TO M.L. D
       MVC 0(2,9),0(8)  M.L. OF D POINTS TO
                      LEFT LINK OF B
```

**END**

**STAR**

Figure TT-10
2.3.3 STRUCTURED TREE DIRECTORY

This section, which includes Random Organization, and the complete File Section, which follows, are an integral part of an information management system.

The directory, files and type of structure, which is chosen for each, will have a great bearing on the speed and efficiency of the system. In the Swanson study, one approach was used. These two parts of the book will allow us to widen our area of selection of approaches we might choose.

We have developed the hierarchic structure concept thus far in the book. Now we will make use of it and others in developing index systems to files for search and retrieval purposes. As with the Swanson study, these methods are slanted toward disk systems.

2.3.3.1 STRUCTURED TREE DICTIONARY

It is important to note that each node (leaf) represents a complete record in the system. The use of three letters for a key is used here as an example. Actually a three-level tree can accommodate several million keys, if there are enough characters in the addressing method. Addressing method here means the KEY/ADDRESS/LIST LENGTH, which amounts to a multiword method.

The first level of the tree is maintained in core storage, and the other two levels are maintained in auxiliary storage. A typical disk notation is used in which T12 means track 2 of disk 1.

The key fragments across the bottom of the tree stand for complete names, but were truncated in order to form a fixed length key
format. This fixed length format could be imposed by the hardware or could be the choice of the programmer. Any number of letters could be used, but as mentioned earlier, the number of letters in the key affects the number of records possible. Three were used here for the sake of brevity.

The search format for the tree is the one mentioned above:

NAME/ADDRESS/LIST LENGTH

The name, address and list length exist in auxiliary storage, and the addressing scheme proceeds from low to high as do the IBM 360 computers.

Level 1 resides in core storage, and it directs us to level 2, which resides in auxiliary storage. This address at level 2 is a track address. The 0 denotes that it is track 0 and 1 denotes that cylinder 1 is used. The highest alphabetic letter here is denoted by COW/T12/*, which signifies that COW marks an upper limit for this section of the tree. The asterisk denotes that there is no list length here, since the branch is not made to the actual list as yet. Level 2 then takes us to T12, which resides on level 3. This level contains the actual lists. T12 takes us to track 2 of cylinder 1. This level contains COW, which is the name part of the address we want. The actual address and length of the record is given here.

The process of decoding this type of dictionary proceeds by the following manner. For example, assume that we have the word COWBOY. The first three letters will be truncated (COW). This key is then compared to the dictionary file in core storage. It is smaller than HOT so COW is selected. This key takes us to track T10, track 0, disk 1, and then proceeds to compare at this level. COW is larger than CAM,
but it does match COW.

This causes a branch to track 2 of disk 1 (T12), where a match is made. The link address takes us to the actual address of COWBOY, and we are given the length of the record that resides at that location.

![Diagram of file organization]

Figure STD-1

2.4 FILES

The file organization can be divided into three major groups:

1. Sequential File Organization
2. Inverted Files (or inverted lists)
3. String structures

The other types of organizational methods listed in this section
can be considered to be combinations of the basic three methods.

A slight emphasis is placed on the Inverted File for reasons of comparison to the RPG methods which occur later in the book.

Figure F.1 depicts the structural organization from the most common at the base, Sequential Organization, to the more sophisticated and complex at the top.

The idea of file partitioning is introduced in this section. The partitioning concept is intended to make access to the disk records faster. The two basic methods of partitioning we will use are Inverted Lists and Multiple Threaded Lists. Finally, we will introduce combinations of these methods, and develop a Page Partition Multilist.

![Diagram](image)

Figure F-1

The Inverted File and the Multilist exist at the extremes of a continuum as depicted by Figure F.1. They represent opposing or contrasting methods of file organization for search and retrieval purposes. The other methods represented here were developed as faster methods were needed. The Multilist and Page Partitioned Multilist evolved as
a result of the use of file structures in the auxiliary and core storage. As will be seen in a later part of the book, the Virtual Memory techniques evolved at a later date.

2.4.1 SEQUENTIAL FILES

Sequential data organization is the most common type used. For example, an inventory file is made up of many records, and each record in turn contains many fields:

- Stock number
- Unit of measure
- On-hand quantity
- Material description
- On-order quantity

The relationship of these fields of the record is that they do relate to a particular stock item. An inventory record for a hand saw
would have a stock number for hand saws, a unit of measure, a material
description of hand saws, and a measure for which the information
about on-hand, on-order and reorder quantities for hand saws if kept.

The sequential file has certain advantages in that there is a
fast access for each relationship. There is a disadvantage when a file
is to be searched until a record having a particular key is found. This
requires an examination of the first record, if the key is wrong, the
next record, etc. This process goes on until the correct record is
found. The updating process is also rather difficult with a sequential
file. If the new record is shorter or longer than the original record
in the file, the adjacent records may be affected or destroyed. The
update process then becomes very costly when there is only one record
to update. The update process is generally used when there are a number
of records to update.

If we have a file of records, it becomes necessary to discover
certain features that exist in the file. For example, if a saw is sold,
the inventory record for saws needs to be isolated so that the value of
the on-hand quantity field can be reduced.

The common attribute here is called a key. By selecting a dif-
ferent key for a file and sorting on the basis of that particular key,
the sequence of records in the file may change, but we can obtain the
desired information.

It is not necessary to store records with keys. There are
times when the sequential arrangement is based solely on the arrival
within the system.
Both these methods can be used for the string structure of a file.

1. Compare them to the Swanson Study.

2. Compare them to the linkages in Section II of Data Structures.

3. Where do the actual linkage processes take place, in core or on disk?
2.4.2 INVERTED FILES

The above example IF-1 is a drawing of the Inverted File structure taken to its ultimate, because the list has a length of 1. This type of structure requires the index to take up as much or more room as the list. The pointer AD(1,1) is a disk address to take us to track 1 of cylinder 1 to a particular location, which has a list length of 1.

A comparison could be made between State and City to determine which ones belong together. This could be done without linking to the list, which means a comparison of the indexes.

The main disadvantage to this method is the large directories needed. It works best as a partially inverted file combined with a random method, such as a hierarchical tree method or sequential method. This makes it possible to invert on just a few keys.
2.4.2.1 CREATION, MAINTENANCE and DELETION OF KEYS

If there is a master file, an inverted file may be created from it. This entails searching through the master file, making a directory from the selected keys, and applies to both the inverted file and the multilist file.

Every time a record is added to the master file, it becomes necessary to update the directories it pertains to.

The Inverted File Algorithm:

1. After the record is entered in the master file, prepare it for entry into the inverted file.
2. Assign the auxiliary address AAD, which may entail some form of the list of available space.
3. Key n in the directory is decoded to the variable length inverted file index.
4. Place Key n in its proper sequence into the directory; if the space is exhausted, use a link to the next page or block.
5. Add one to the list length in the index.
6. Continue steps 3 to 5 for all keys of the new record.
7. Store the new auxiliary address at AAD.

Deletion of Keys

The deletion of keys causes no problem, as far as overflow is concerned, since there is a reduction of space being used.

The Deletion algorithm:

1. Retrieve record from auxiliary storage and delete Key n.
2. Decode Key n in the proper directory of the inverted file list.

3. Delete the address from the list (this address is the address AAD residing in the inverted file list).

4. Repeat steps 2 and 3 until all keys of the record have been deleted that are required to be deleted.

Deletion of Records

The Record Deletion algorithm:

1. Transfer record from auxiliary storage location to core.
2. Set the record delete bit.
3. Decode every key of the record in the index and remove the record address (AAD) from all inverted file lists.
4. Decrement each file list, which has been affected by 1.

Addition of Keys

A problem arises here pertaining to record length. After the addition of a key, it is possible that the track could overflow. If so, the record is deleted and a link is inserted to another track where this whole record is inserted.

The Addition of Keys algorithm:

1. Decode the Key n in the proper directory to the variable length inverted file.
2. Determine the proper sequence, insert AAD of the record of Key A, and add 1 to the list length.
3. Continue steps 1 and 2 for all keys to be added.
4. Transfer the record from AAD (auxiliary address), and
add new keys to record.

5. If the record doesn’t cause overflow of the track after packing it, return it to AAD.

6. If the record causes overflow, insert a link to another track, and place the whole record there.

2.4.2.2 LOGICAL OPERATOR "OR"

The logical operator "OR" effects a union of Directory A and Directory B. A key that resides in either Directory A or Directory B will be included in Directory C, and if the key appears in both directories, it appears only once in Directory C.

\[ C = A \lor B \]

The Logical Operator "OR" algorithm:

1. Set 3 index registers (one register each for A, B, and C) to 1.

2. Compare unit 1 of A to unit 1 of B (the keys). If the key of A is smaller than the key of B, move the key of A to Directory C and increment index A and index C. If the key of A is equal to the key of B, move the key of A to Directory C and increment index A and index C. If the key of A is larger than the key of B, move the key of A to Directory C and increment index A and index C.

3. Repeat step 2 until Directories A and B have both been exhausted.

4. Stop.
\[ C = A \text{ "OR" } B \]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GO</strong></td>
<td><code>CIC 0(2, 9), 0(18)</code> <strong>COMPARE FILE A TO FILE B</strong></td>
</tr>
<tr>
<td><code>BC</code></td>
<td><code>4, ALO</code> <strong>A IS LOW</strong></td>
</tr>
<tr>
<td><code>BC</code></td>
<td><code>E, EQUAL</code> <strong>BRANCH IF EQUAL</strong></td>
</tr>
<tr>
<td><code>MVC</code></td>
<td><code>0(2, 11), 0(18)</code> <strong>B IS LOW, MOVE FILE B TO</strong></td>
</tr>
<tr>
<td><code>AR</code></td>
<td><code>10, HR</code> <strong>INCREMENT RIO 0(B)</strong></td>
</tr>
<tr>
<td><code>BC</code></td>
<td><code>15, MNC</code> <strong>INCREMENT FILE C</strong></td>
</tr>
<tr>
<td><strong>ALO</strong></td>
<td><code>MVC 0(2, 11), 0(9)</code> <strong>MOVE FILE A TO C</strong></td>
</tr>
<tr>
<td><code>AH</code></td>
<td><code>9, HR</code> <strong>INCREMENT REG 9, FILE A</strong></td>
</tr>
<tr>
<td><code>CH</code></td>
<td><code>REN, 0(9)</code> <strong>IS FILE A FINISHED</strong></td>
</tr>
<tr>
<td><code>BC</code></td>
<td><code>R, OUT</code> <strong>YES BRANCH OUT OF SUBROUTINE</strong></td>
</tr>
<tr>
<td><code>INCC</code></td>
<td><code>HR 11, M2</code> <strong>INCREMENT II = FILE C</strong></td>
</tr>
<tr>
<td><code>BC</code></td>
<td><code>15, GO</code> <strong>BRANCH BACK TO GO</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MVC</code></td>
<td><code>0(2, 11), 0(10)</code> <strong>MOVE FILE B TO FILE C</strong></td>
</tr>
<tr>
<td><code>AH</code></td>
<td><code>10, HR</code> <strong>INCREMENT REG 10 = FILE B</strong></td>
</tr>
<tr>
<td><code>MVC</code></td>
<td><code>0(2, 11), 0(9)</code> <strong>MOVE FILE A TO FILE C</strong></td>
</tr>
<tr>
<td><code>AH</code></td>
<td><code>9, HR</code> <strong>INCREMENT REGISTER 9</strong></td>
</tr>
<tr>
<td><code>CH</code></td>
<td><code>REN</code> <strong>IS FILE</strong></td>
</tr>
<tr>
<td><code>BC</code></td>
<td><code>R, OUT</code></td>
</tr>
<tr>
<td><code>BC</code></td>
<td><code>15, INCC</code> <strong>INCREMENT FILE C</strong></td>
</tr>
<tr>
<td><code>OUT</code></td>
<td><code>D.CR 15, 14</code> <strong>BRANCH BACK TO MAIN PROG.</strong></td>
</tr>
<tr>
<td><code>END</code></td>
<td><code>STAR</code></td>
</tr>
</tbody>
</table>

**Figure IF-3**
2.4.2.2.2 A AND NOT B

Directory C will contain only the keys of Directory A that are not contained in Directory B.

The algorithm for C = A "AND NOT" B

1. Set 3 index registers (one each) for A, B and C to 1.
2. If the key of A is smaller than the key of B, move the key of A to Directory C, increment indexes A and C.
   If the key of A is greater than the key of B, increment the index of B.
3. Repeat step 2 until Directories A and B have been exhausted.
4. Stop.

\[ C = A \text{ "AND NOT" B} \]

\[
\begin{array}{ccc}
B & A & C \\
2 & 10 & 11 \\
4 & 11 & 12 \\
5 & 12 & 40 \\
10 & 20 & 40 \\
15 & 40 & 42 \\
20 & & \\
42 & & \\
\end{array}
\]

Figure IF-4
### Figure IF-5

<table>
<thead>
<tr>
<th>Statement</th>
<th>Function</th>
<th>Memory Reference</th>
<th>Comment</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STAR</strong></td>
<td>Ah</td>
<td>9, AAA</td>
<td>LOAD ADDRESS OF A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ah</td>
<td>18, BBB</td>
<td>LOAD ADDRESS OF B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ah</td>
<td>11, CCC</td>
<td>LOAD ADDRESS OF C</td>
<td></td>
</tr>
<tr>
<td><strong>GO</strong></td>
<td>Cl, d</td>
<td>0(2, 9), c(12)</td>
<td>COMPARE FILE A &amp; B</td>
<td></td>
</tr>
<tr>
<td><strong>BC</strong></td>
<td>7, ALD</td>
<td>IF A IS LOW, BRANCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ah</td>
<td>9, HR</td>
<td>EQUAL = 50, GO TO NEXT A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ch</td>
<td>9, REND</td>
<td>IS FILE Ainished</td>
<td></td>
</tr>
<tr>
<td><strong>BC</strong></td>
<td>8, OUT</td>
<td>BRANCH TO OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BC</strong></td>
<td>15, GO</td>
<td>BRANCH BACK TO GO</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ALD</strong></td>
<td>Ah</td>
<td>10, HR</td>
<td>INCREMENT FILE B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ch</td>
<td>10, REND</td>
<td>IS FILE B finished</td>
<td></td>
</tr>
<tr>
<td><strong>BC</strong></td>
<td>7, GO</td>
<td>NO = BRANCH BACK TO GO</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AVC</strong></td>
<td>0(2, 11), 0(9)</td>
<td>YES = 50, MOVE A TO C</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ah</td>
<td>9, HR</td>
<td>INCREMENT A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ah</td>
<td>11, HR</td>
<td>INCREMENT C</td>
<td></td>
</tr>
<tr>
<td><strong>BC</strong></td>
<td>15, GO</td>
<td>BRANCH BACK TO GO</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OUT</strong></td>
<td>BR R, 15, 1H</td>
<td>RETURN TO MAIN PROGRAM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.4.2.2.3 LOGICAL OPERATOR "AND"

Let us examine the operations it takes the system to create a resultant file, if we take two indexed files and request a retrieval of keys for which both files apply. Let's scan file A and file B.

It is written this way in set theory:

\[ \text{INDEX } C = A \cup B \]

This allows us to retrieve the keys from Directory A that also reside in Directory B.

The algorithm for Logical Operator "AND"

1. Set 3 index registers (one each) for A, B, and C to 1.
2. Compare unit 1 of A to unit 1 of B.
   - If the key of A is equal to the key of B, increment A to the next key of A, and increment B to the next key of B, move A to C and increment C.
   - If the key of A is smaller than the key of B, increment A to the next key of A.
   - If the key of A is larger than the key of B, increment B to the next key of B.
3. Repeat step 2 until all the keys in B have been investigated.
2.4.3 MULTILIST FILE

The use of "cells" or "pages" allows us to use fixed-size blocks to implement a method called the Multilist File. A sequential index contains the key values by which records are indexed. The pointer, which is associated with each key, points to the list of records that has the particular key value in question. The addressing method uses a page number and record number within the page.

A group of records can be retrieved at one time. This implies that access time can be reduced, however the saving is brought about only when a number of records are desired.

See figure M.F.-1, the Multilist File. The index, or directory contains keys L, M, N, and O. If we had a key that started with the letter L, the directory would decode it and take us to (3,3)/11, which translates to page 3, record 3 and tells us that there are 11 records in the list.

The other two methods discussed here use the same records and the same scheme for presenting them. There is a key, an address, and a list length sequence for each system.

The other two methods discussed here are presented with the same type drawing for ease of understanding.

The same logical operations performed on the inverted files can be performed on the multilist files.
Multilist

Figure M-1
Controlled Multilist Lengths

The same pattern is followed here with the exception that the list lengths are not more than four records long. The number in the lists is arbitrary. This avoids the long lists that the Multilist method had.

See Figure M.F.2, the Controlled Multilist File. The index contains the same keys that the Multilist File did, but we notice that there are more divisions within the key group. This time L has three divisions and the same number of records.
Multilist Organization for Pages

The Page Multilist files are linked so that they do not cross page boundaries. The advantage to this is that it does not require several pages of information when a key is used to retrieve certain information. A Multilist File might require several pages to be accessed.

The cells or pages become the partition instrument. This allows the list structure and the partition (page partition) to function.

Figure M-3
2.5 SORTING

The object of the sort section is to present various methods of sorting to allow the reader a wider choice of methods from which to select.

The activity section at the end of this section suggests examples that are contained in the various Basic Assembler Language programs in the book.

SORTING METHODS

RECORD SORT

The sort algorithm is applied directly to the complete record by using the key in the record. This means the complete record is moved when a change is indicated. The end result is that each record is in its proper place when the file is sorted.

2.5.1.1 KEY SORT

The key sort works with the keys only, as opposed to the whole record. This includes an associative table of address pointers. Associative in this instance refers to a memory device, in which each cell contains information pertaining to the key and entry. The entry part tells us whether a cell has an entry pertaining to the key that accessed the particular cell. If there is no entry, this signal is returned as "not found" or something similar.

There are two types of key sorts, the detached and the non-detached key sort.

2.5.1.2 DETACHED KEY SORT

The algorithm for the Detached Key Sort:

The algorithm for the Detached Key Sort:
1. The record keys are placed in an associative table with address pointers.

2. Apply the internal sorting algorithm to the associative table only.

3. If a transfer is to be made, make it include only the key and associated record address.

4. When the associative table is sorted, the complete records in the file are moved to output according to the sorted order of the address pointers.

This algorithm is quite forward, if we separate the key and address from the rest of the record to form a new table.

2.5.1.3 THE NONDETACHED KEY SORT

The nondetached key sort forms a table which contains pointers to the keys of the associated records. This method applies the sort algorithm indirectly through the associated address pointers in the table. The address pointers (not the keys) are moved during the sort. Both the table of addresses and the stored file are referenced during the sorting procedure.

When the sorting algorithm is finished, only the record address pointers have been sorted. The retrieval process requires that the sequence of the record address associative table be followed.

2.5.2 RADIX SORT

This method is used for sorting punched cards mechanically, but can be used for the computer as well.

The least significant digit of each key is examined. Then the
record is assigned to a pocket which depends on the value of the digit. After all the records have been examined, the records are distributed again, and so on, until all the digits of the key have been used for distribution. After the collection of the pass following the most significant digit, the records are in order. For decimal keys, ten pockets are needed, or else each pass could be replaced by two passes, in which case fewer pockets would be required.

The total number of passes is equal to the number of digits in the key, and the capacity of each pocket must be sufficient for all the records that might end up there.
Radix Sort

|   |   | 19 | 38 | 72 | 56 | 66 | 13 | 57 | 13 | 40 | 23 | 71 | 2 | 45 | 15 | 07 | 23 |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

| 77 | 28 | 57 | 66 | 45 | 94 | 49 | 03 | 23 | 19 | 72 | 91 | 70 |

| 71 | 38 | 22 | 57 | 56 | 66 | 45 | 24 | 74 | 13 | 23 | 03 | 03 | 72 | 91 | 40 |

| 71 | 94 | 72 | 66 | 57 | 45 | 38 | 23 | 13 | 03 |

| 74 | 91 | 87 | 72 | 16 | 57 | 56 | 45 | 40 | 23 | 23 | 15 | 03 |

Figure RS-1
2.5.3 QUICKSORT

The quicksort algorithm can be classified as a partitioning type algorithm, and it was developed by Hoare. The main aspects of the algorithm are:

1. The first pass located the item that occupies the middle spot of the list and classifies it as a bound.

2. This item is copied into a temporary location and replaced with the bottom of the list.

3. The list is scanned from the top, and each item is compared in sequence to the bound until an item is found that is larger than, or equal to, the bound.

4. If the new item is larger than the bound item, the new item is moved to the bottom of the list, creating a vacant spot in the top part of the list.

5. The top-down scan is then continued, alternating the scan each time a transfer is made, until all the items have been scanned. When the two top pointers coincide, the list will be partitioned into two parts.

6. The count is then placed in the vacant spot between the two partitions. This is the spot where the two pointers coincide. It now occupies its sorted position, because all items above it are smaller, and all items below it are larger or equal to it.

The partition that has the most elements is then stacked for later processing, while the algorithm proceeds to partition the smaller list in order. This continues until each item has been placed into its
proper sorted position either by being selected as a bound or by remaining a single item in a partition.

2.5.4 TOPOLOGICAL SORT

The topological sort is used in situations that require partial ordering. Pert and Critical Path methods are good examples of this process. Partial ordering occurs in mathematics in situations such as $A = B$ (between real numbers), and also between $A$ and $B$ as $A \leq B$ in set theory.

The critical path program presented in a previous chapter has many good examples of the topological sort. Presented here is the algorithm (in part).

5. Sort the table on the basis of lowest starting event number.
6. Locate activity of the lowest starting event number.
7. Locate and place in table TT0 all activities with end event numbers that match the start events of (6).
8. Sort TTWO on the basis of ending times.
9. Subtract ending time of each activity from the ending time of the longest activity in table TTWO.
10. Enter difference in slack time position.
11. Enter longest event time from table TTWO into start time for activity (6).
12. Compute end time for activity (6).
13. Index activity (6).
14. Transfer table TTWO and activity (6) back to the main table.
15. Test for last event.
16. NO—back to (6).

2.5.5 QUADRATIC SORT

The quadratic sort divides the array into four equal parts and then searches each of the four groups for the smallest data. This number is erased and moved to a temporary area, where each of four are compared. The smallest of these is transferred to the output array. The process, with the exception of the division into four units, is combined until all the original array is blank, and all the intermediate
temporary areas are blank. The final result is a sequential ordering in the output array.

2.5.6 CALCULATED ADDRESS SORT

Addresses are calculated by one of the random methods presented earlier. If the cell is vacant, the data is placed there, if the cell is occupied, a compare is made, and the smaller of the two is inserted into the cell, and the larger goes to the next cell, after all the following data has been moved down one cell sequentially.
2.5.7 SORTING BY INSERTION

Each key is examined in turn and inserted into the correct place. The earlier members are pushed down when the need arises. The mechanics of moving the earlier records down is also used as a method of insertion with lists. The actual number is n/4. This method of sorting is useful when there isn't much core storage.
### Sorting by Insertion

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T.A.B.</td>
<td>CP, TAB, TAB, T3</td>
</tr>
<tr>
<td></td>
<td>SC, 10, DWN</td>
</tr>
<tr>
<td></td>
<td>SC, 15, TAB</td>
</tr>
<tr>
<td></td>
<td>UNCONDITIONAL BRANCH BACK TO TAB</td>
</tr>
</tbody>
</table>
2.5.8 INTERCHANGE SORT

Figure IS-1
**Interchange Sort**

### LOOP
- **CP**: 16 (2, 8), 16 (2, 5)
- **BC**: 13, COMP
- **OT**: SW, X'01'

### MVC
- **BN**: 30 (S0), 30 (E)
- **BIN**: 30 (S0), 30 (E)

### COMP
- **AH**: 8, TNY
- **CLC**: 30 (R, 8), STAR
- **BC**: 7, LOOP
- **TM**: SW, X'01'
- **BC**: 2, SORT

---

**Figure M-2**
2.6 MERGING

2.6.1 TWO WAY MERGE

Many different types of sorting are descendants of this basic approach to the merge. The two way merge compares pairs of keys, and each pair, of which the smaller key is placed first. After one pass, the initial group \( N \) consists of \( N/2 \) strings of length two. These pairs of strings are combined to form \( N/4 \) strings of length four. These pairs are combined to form \( N/8 \) strings, until one string results. If \( N \) is of the form \( 2^p \), it will require \( p \) passes to complete the sort.
Merging Two Sorted Files

The comparison is made between the keys in the records. A merge can be made with the order ascending or descending. For our example, we will use the ascending merge, since our two input files are sorted in ascending order.

This merge effects the union of File A and File B into File C. If a number appears in both files, the number in File A will receive first precedence.

The Merge Algorithm

1. Set two indexes (1 each) for A and B, also set an index for C. (Set these indexes to 1.)

2. Compare the key of File A to the key of File B, if the key of A is smaller than the key of B, move the key of A to File C and increment index A and index C.
   a. If the key of A is equal to the key of B, move the key of A to File C.
   b. If the key of File A is larger than the key of File B, move the key of B into File C, and increment index B and index C.

3. Repeat step 2 until all of Files A and B have both been exhausted.
2.6.1.2 MERGING ORDERED FILES WITH SUBFILES

There are four possibilities that can exist here:

1. There can be no problem, such as the preceding example, where no changes are necessary.

2. A step down condition can exist (a single step down).

3. A double step down condition can exist.

4. It may be necessary to perform a roll out.

These 3 numbers will roll out when File A has finished.
The single step down is given in the example above, M. The number 3 in File B is a step down, because it is smaller than the number preceding it in the file, and it is also smaller than the number that follows it in File B. This means that File B will not be used again until File A has a step down also. The sequential process continues with File A until A results in the step down mentioned above. If there are no further step downs in File A, then File B will continue sequentially. This is called a roll out.
Merging Ordered Files with Subfiles

<table>
<thead>
<tr>
<th>STAR</th>
<th>L.H.</th>
<th>9, A</th>
<th>LOAD ADDRESS OF FILE A</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.H.</td>
<td>10, 388</td>
<td>LOAD ADDRESS OF FILE B</td>
<td></td>
</tr>
<tr>
<td>L.H.</td>
<td>11, 222</td>
<td>LOAD ADDRESS OF FILE C</td>
<td></td>
</tr>
</tbody>
</table>

**G03**

<table>
<thead>
<tr>
<th>MVC</th>
<th>TEMP, O(9)</th>
<th>MOVE 9-Z TO COMPAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>AH</td>
<td>9, H2</td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>9, TEMP</td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>9, 8, 4</td>
<td>BRANCH TO SET SWITCH</td>
</tr>
<tr>
<td>SH</td>
<td>9, H2</td>
<td>RETURN 9 TO ORIGINAL NR</td>
</tr>
</tbody>
</table>

**G04**

<table>
<thead>
<tr>
<th>MVC</th>
<th>TEMP, O(12)</th>
<th>MOVE 10-Z TO FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AH</td>
<td>10, H4</td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>10, TEMP</td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>10, 8, 4</td>
<td>BRANCH TO SET SWITCH</td>
</tr>
<tr>
<td>SH</td>
<td>10, H2</td>
<td></td>
</tr>
</tbody>
</table>

**Table**

<table>
<thead>
<tr>
<th>L.H.</th>
<th>16</th>
<th>30</th>
<th>60</th>
<th>90</th>
<th>120</th>
<th>150</th>
<th>180</th>
<th>210</th>
<th>240</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVC</td>
<td>O(2, 11), O(10)</td>
<td>COMPARE FILE A TO FILE B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AH</td>
<td>10, H2</td>
<td>INCREMENT FILE A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>15, 803</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure M-6**
### Merging Ordered Files with Subfiles

#### Figure M-6 (Part-2)

<table>
<thead>
<tr>
<th>COL</th>
<th>MVC</th>
<th>0(g, 11), 0(8)</th>
<th>MOVE FILE A TO FILE C.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AH</td>
<td>11, M2</td>
<td>INCREMENT FILE C.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AH</td>
<td>9, M2</td>
<td>INCREMENT FILE A.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVC</td>
<td>TEMP, 0(3)</td>
<td>MOVE FILE A TO TEMP.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>9, TEMP</td>
<td>INCREMENT FILE A.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td>4, ROL</td>
<td>IF NH IS LOW, BRANCH TO ROL.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>9, END</td>
<td>IS IT THE END OF FILE A?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVC</td>
<td>SW, ONE</td>
<td>YES? SET SWITCH ONE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td>15, COMP</td>
<td>BRANCH TO COMP.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COL</th>
<th>MVC</th>
<th>0(g, 11), 0(10)</th>
<th>MOVE FILE B TO FILE C.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AH</td>
<td>9, M2</td>
<td>INCREMENT FILE C.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AH</td>
<td>10, M2</td>
<td>INCREMENT FILE B.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVC</td>
<td>TEMP, 0(3)</td>
<td>MOVE AREA SUBDIVIDED BY 10 TO TEMP.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>9, TEMP</td>
<td>INCREMENT FILE B.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td>4, ROL</td>
<td>IF NH LOW = BRANCH TO ROL.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>9, END</td>
<td>END OF FILE B?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td>4, ROL</td>
<td>NO = BRANCH BACK TO ROL.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVC</td>
<td>SW, ONE</td>
<td>YES? SET SWITCH 1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td>15, COMP</td>
<td>BRANCH TO COMP.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure M-6 (Part-2)**
Double Step-Down

The double step-down is treated as if there were no step down at all. In fact it is handled like the straight merge.

![Diagram showing step-down process]

Figure M-7

The double step-down is a step down which occurs in both files, so it operates in the same manner as a merge.

Activities

1. What type of sort does the construction program use? If the reader will notice, the sort flow chart was taken out of context for emphasis.

2. What type of sort does the STAT program use? Compare it to the example presented in this chapter.

   (Answer) A displacement insertion type.

3. What type of sort does cause problems when there are many
digits to sort?

(Answer) Interchange Sort.
Merging—Double Step-Down

**Figure M-8**

<table>
<thead>
<tr>
<th>Command</th>
<th>Register</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.H</td>
<td>9, A A A</td>
<td>LOAD</td>
<td>START OF FILE A</td>
</tr>
<tr>
<td>L.H</td>
<td>10, B B B</td>
<td>LOAD</td>
<td>START OF FILE B</td>
</tr>
<tr>
<td>L.H</td>
<td>11, C C C</td>
<td>LOAD</td>
<td>START OF FILE C</td>
</tr>
<tr>
<td>C.S.</td>
<td>A Y C.</td>
<td>TEMP, 0.(10)</td>
<td>MOVE A TO TEMP</td>
</tr>
<tr>
<td>C.H</td>
<td>9, I H I</td>
<td>INCREMENT</td>
<td>A</td>
</tr>
<tr>
<td>B.C.</td>
<td>4, S W H</td>
<td>COMPARE A TO A + Z</td>
<td></td>
</tr>
<tr>
<td>S.H.</td>
<td>9, N Z</td>
<td>SET A</td>
<td>BACK 2 BYTES</td>
</tr>
<tr>
<td>G.0.</td>
<td>M Y C.</td>
<td>TEMP, 0.(10)</td>
<td>MOVE B TO TEMP</td>
</tr>
<tr>
<td>A.H.</td>
<td>10, I H I</td>
<td>INCREMENT</td>
<td>B</td>
</tr>
<tr>
<td>C.H.</td>
<td>10, TEMP</td>
<td>COMPARE B TO P + Z</td>
<td></td>
</tr>
<tr>
<td>B.C.</td>
<td>4, S W H</td>
<td>STEP DOWN</td>
<td>IF B EQ. LOW</td>
</tr>
<tr>
<td>S.H.</td>
<td>10, N Z</td>
<td>SET FILE B</td>
<td>BACK 2 BYTES</td>
</tr>
<tr>
<td>B.C.</td>
<td>15, C O M</td>
<td>BRANCH TO</td>
<td>MOVISEC</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Command</th>
<th>Register</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G.0.</td>
<td>M Y C.</td>
<td>0.(2, 11), 0.(10)</td>
<td>COMARE FILE A TO FILE B</td>
</tr>
<tr>
<td>B.C.</td>
<td>9, E G A</td>
<td>9, A D</td>
<td>BRANCH TO EQ.</td>
</tr>
<tr>
<td>B.C.</td>
<td>9, A D</td>
<td>9, A D</td>
<td>BRANCH TO EQ.</td>
</tr>
<tr>
<td>A.Y C.</td>
<td>0.(2, 11), 0.(10)</td>
<td>MOVE FILE A TO FILE B</td>
<td></td>
</tr>
<tr>
<td>A.H.</td>
<td>9, I H I</td>
<td>INCREMENT</td>
<td>FILE C</td>
</tr>
<tr>
<td>B.C.</td>
<td>15, I N C</td>
<td>INCREMENT</td>
<td>FILE C</td>
</tr>
<tr>
<td>A.H.</td>
<td>11, I H I</td>
<td>INCREMENT</td>
<td>FILE C</td>
</tr>
<tr>
<td>B.C.</td>
<td>15, 6 0 3</td>
<td>BRANCH TO MOVISEC</td>
<td></td>
</tr>
<tr>
<td>E.G.A.</td>
<td>M Y C.</td>
<td>0.(2, 11), 0.(10)</td>
<td>MOVE FILE A TO FILE C</td>
</tr>
<tr>
<td>A.H.</td>
<td>10, I H I</td>
<td>INCREMENT</td>
<td>FILE C</td>
</tr>
<tr>
<td>A.Y C.</td>
<td>0.(2, 11), 0.(10)</td>
<td>MOVE FILE A TO FILE C</td>
<td></td>
</tr>
<tr>
<td>A.H.</td>
<td>9, I H I</td>
<td>INCREMENT</td>
<td>FILE C</td>
</tr>
<tr>
<td>C.H.</td>
<td>9, I N D</td>
<td>BRANCH TO SET SWITCH</td>
<td></td>
</tr>
<tr>
<td>B.C.</td>
<td>6, A A A</td>
<td>BRANCH TO SET SWITCH</td>
<td></td>
</tr>
<tr>
<td>B.C.</td>
<td>6, A A A</td>
<td>BRANCH TO SET SWITCH</td>
<td></td>
</tr>
</tbody>
</table>

---

**Figure M-8**
Merging—Double Step-Down

Figure M-8 (Part-2)
2.7 SEARCH

Search structures are concerned with the retrieval of data from data structures. The key plays an important part, because the key is the part of the record which is searched. Frequently, large data structures must be processed and tabulated.

A search structure consists of a search process together with a data structure method. Generally, it takes work on both of these, if either one is affected or improved; therefore it is quite important for the programmer to be cognizant of the system as a whole, not just the search, but the search and the structure also.

The selection of a search structure does in fact determine the search and insert and delete methods for a particular data base.

The speeds of these methods are also an important determining factor for selecting a search structure.

The linear search, as presented with the sequential file previously, examines each key in sequence and is terminated with a matching key, or when the last item in a file matches, a termination is effected.

Generally the number of items tested depends on the number of items in the complete data base. The number of items actually tested during a linear search is estimated at $N/2$.

2.7.1 LINEAR SEARCH

A search is made of the array $TH$ of $N$ integers for one that is equal to $Q$.

2.7.2 BINARY SEARCH

The binary search provides a fast search process through the
ordering of the data base, however the insert and delete process is relatively slow. The continual ordering and reordering of the data base each time a new item is added or taken out, is relatively slow.

![Diagram]

Figure BS-1

The search starts by entering the file or list near the middle. The word that divides the list in two is sometimes called a fence (a form of partitioning). By comparing a fence key with the key of the desired record, a decision is made whether to look further in the sublist above or below. If the sublist above is taken, the list is divided by 2, \((X/2)\), and another fence is created at this location. This process continues until either a list top, a list bottom, or matching record is determined.
Binary Search

Figure BS-2

Table = First address of table

L = Length of entry in table

P = Pointer

I = Iteration
## Binary Search

<table>
<thead>
<tr>
<th>FORTRAN STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>I = N - N/2</td>
</tr>
<tr>
<td>IND = I - 2/2</td>
</tr>
<tr>
<td>GO TO 2A1</td>
</tr>
<tr>
<td>100 I = I + IND</td>
</tr>
<tr>
<td>110 IF (IND &lt; 1) 120, 140, 120</td>
</tr>
<tr>
<td>120 THDS = IND - THDS/A</td>
</tr>
<tr>
<td>2A1 IF (X - A(I)) 150, 160, 160</td>
</tr>
<tr>
<td>130 I = I - IND</td>
</tr>
<tr>
<td>140 IF (X = A(I)) 150, 160, 160</td>
</tr>
<tr>
<td>150 CONTINUE IF NO ELEMENT OF A = X</td>
</tr>
<tr>
<td>160 CONTINUE IF A(I) = X</td>
</tr>
</tbody>
</table>

Figure BS-3
Table Search Program

<table>
<thead>
<tr>
<th>FORTRAN STATEMENT</th>
<th>LEN(S)</th>
<th>LEN(Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIMENSION TABLE (200)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ 70 TABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70 FORMAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ 80 AMNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80 FORMAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90 DO 100 I = N, 1, 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF (AMNT = TABLE(I), 100, 200, 100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 CONTINUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF (I = 10) / 10, 150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150 PRINT 100, AMNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200 PRINT 250, J, I, AMNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>250 FORMAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J = J + 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N = J + 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GO TO 90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure TS-1
2.8 DYNAMIC STORAGE ALLOCATION

The list processing languages have a common feature, which is, memory space for data structures does not have to be preassigned. The storage for each structure is allocated as it is needed and usually not sequentially. This is accomplished by the linking process.

In order to be able to reassign the use of memory cells during execution of a list processing program, a list processing language must provide for:

1. A storage of cells available for use.
2. Systems for obtaining "new" cells from, and returning unneeded cells to the store.

We will see that this also applies to virtual memory, which follows this section. Knuth mentions two methods for storage allocation but considers the First Fit Method the best.

![Diagram of storage allocation](image-url)
Dynamic Storage Allocation

Figure DS-2
Dynamic Storage Allocation

Figure DS-2
2.9 VIRTUAL MEMORY

This discussion is based on a paper by Denning,¹ who has made a survey of virtual memory.

Earlier developments brought the state of the art to a point where computer costs would be too great to provide the large memories that were being suggested.

The proposal for the Atlas computer in 1961 set forth a one level store memory. It was known as Virtual Memory. The central idea is that "address" is entirely different from "physical location." The computer hardware and software automatically transfer information into memory at the precise time it is needed for processing. Also the hardware and software arrange for the program-generated addresses to be directed to memory locations that contain the information addressed.

Virtual memory presented a great potential for overcoming part of the problems of storage allocation, because the memory use was based on system-observed actual use of space.

The mechanisms for affecting virtual memory now become quite important. The mechanisms referred to here are Segmentation and Paging.

Segmentation organizes address space (not memory) into variable size segments, while Paging organizes address space into fixed size pages of contiguous addresses.

The programmer is allowed to work with addresses which are different from memory, then the system provides the mechanism for translation of these program-generated addresses into the proper memory

location addresses. The programmer uses addresses called "names" or "virtual addresses".

This set of names is called a name space or address space, while the address used by memory is "location" or "memory address".

Since these two types of addresses are different, it requires a mapping system to compact the larger system address space into the smaller memory space.

Address Translation Mechanism

The addressing scheme allows the programmer to use a two component address technique which is \((s, w)\). The "\(s\)" is the segment and the "\(w\)" is the word name which resides within the segment.

The segment is loaded into a contiguous area of memory at base address \(a\). The letter \(b\) designates the number of locations \(s\) occupies.

Each entry segment is called a descriptor; the \(s\)th descriptor contains the base limit information \((a, b)\) for segment \(s\) if \(s\) is present.
in memory, and is blank if it is not present.

Segment Table

Paging divides memory into equal size blocks of locations. "Page Frames" function as sides of residence for matching size blocks of virtual addresses. A Page serves dual functions:

1. Unit of information storage.
2. Transfer between main and auxiliary storage.

Page Frames are identified by a "frame address" (the location of first word of the page frame). The addresses for pages are written (p, w) where p is a page number and w is word number contained in page p.
Page size was found to work best at forty-five words per page. Segmentation and Paging can be combined, if computer systems have a good selection of register to register operations.

2.3.3.2 RANDOM ORGANIZATION

The Random Organization data structure is based on the principle of retrieving and storing records on the basis of a predictable relationship between a key of the record and the address of the location the record is stored in storage media.

This random organization is used in information retrieval and symbol tables, which is to say, a directory or dictionary type process. One method of developing a directory or dictionary type process, is to use the alphabet as an array in which pointers link the directory to the various addresses which partition storage.
The relationship represented here is the binary code of the alphabet letters themselves. The subtables can be chained together. This random organization method has had various names, such as computed table, hash table, key transformation table, but they all achieve the predictable relationship between the key of the record and the location it is to be stored in or retrieved from. The alphabetic method above in Figure RO-1 is an example of the directory look up method. The key of the record was compared to the directory and compare was equal; the direct core address, and the key were brought together.

Another method of dividing the directory is to divide it into ten tables. This method is used frequently with the linear search. This can be done without any ordering of data because the size of the tables will be relatively small. The key is divided by ten, and the remainder (0–9) is used to compute the location to store the data in

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>AAA</td>
</tr>
<tr>
<td>B</td>
<td>BAA</td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>BAB</td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>COY</td>
</tr>
<tr>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>

Figure RO-1
one of the ten tables. This is sometimes called a hash table.

Another Random Organization method is to choose some of the bits from the middle of the square of the key. This means choose enough bits to be used as an index to determine the address of any item in the table. Since the square depends on all of the bits of the key, different keys will give rise to different hash addresses in the hash table.

When keys are multiword items such as NAME/ADDRESS/LIST LENGTH, it is possible to take the sum of the bits and transform them into an address or location. Also multiplication and division can be performed on parts of the key, but it is important that the calculation does not come out to zero a good percentage of the time.

Another way of presenting the random address method is to cut the key up into n-bit sections, where n is the number of bits needed for the hash address, and then to form the sum of all the sections. The low order n bits of the sum is used as the calculated address. This method can be used for both single-word keys and multi-word keys.
SECTION III

FORTRAN
III FORTRAN

The Fortran section of this book is slanted more toward the programming proficiency required for the DPMA test; however it also includes some Basic Assembler Language examples for comparison.

The first part of the section presents a little chart for the statements that exist in Fortran I, II, and IV. The following material is pertinent to the type required for the test. The remaining material is presented in the form of questions, with answers given.

It is assumed that the reader has a working knowledge of Fortran (at least to the degree that the basic statements are understood). If the reader does not have this facility as yet, he can still develop the proficiency to pass the DPMA test by analyzing the small programs given herein.

III FORTRAN

1. Introduction
2. Summary of Statements that Exist in Fortran I, II, and IV
3. Expression Formation
4. Hierarchy of Operation
5. Nested Iterative Input and Output
6. Fortran Shortcuts
7. Buffering with Fortran
8. Syntax
9. Array Search
10. Sort Ascending and Descending
11. Installment Note Program
12. Statistics

12.1 BAL Statistics Program—STAT

13. Selected Fortran Programs in the Form of Answers to Questions

13.1 Compound Interest Rate

13.2 Plotting

13.3 Integration by Simpson's Rule

13.4 Quadratic Equation

13.5 Linear Equation

13.6 Computed Go To

13.7 The Derivative

13.8 Evaluate a Polynomial

13.9 Functions

14. Arithmetic Questions and Answers
2. SUMMARY OF STATEMENTS THAT EXIST IN FORTRAN I, II, AND IV

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>FORTRAN I</th>
<th>II</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT n, list</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Arithmetic Statement $v = a$</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Arithmetic Statement function name $(x, y, z, ..) = h$</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ASSIGN i TO n</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>BACKSPACE i</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>BLOCK DATA</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CALL name $(h, i, j, ..)$</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CALL SWITCH $(n, i,)$</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>COMMON $(h, i, j, ..)$</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>COMPLEX</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DATA $ijk.../ e, f, g.../o, p, q, .../$</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DEFINE DISK $(n,m)$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIMENSION $u(i), v(i),..$</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DO $n i = j, k, m$</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DOUBLE PRECISION $d, e, f,...$</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>END</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>END FILE i</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EQUIVALENCE $(i, j, k, ..), (x, y, z, ..)$</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>EXTERNAL $e, f, g,...$</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>FETCH $(i) $d, e, f,...$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIND $(i)$</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORMAT $(s, s, s, ....)$</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>STATEMENT</td>
<td>FORTRAN</td>
<td>I</td>
<td>II</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>---------</td>
<td>---</td>
<td>----</td>
</tr>
<tr>
<td>FUNCTION name (i, j, k, ...)</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GO TO n, (i, j, k, ...)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>GO TO (j, k, l, ...), i</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GO TO n</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IF (x) i, j, k (arithmetic)</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IF (sense switch n) i, j</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>IF (x) s (Logical)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER a, b, c,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOGICAL a, b, c, ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical statement v = x</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PAUSE</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PRINT n, list</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PRINT n, iterative list</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUNCH n, list</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUNCH n, iterative list</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>READ n, list</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>READ n, iterative list</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>READ INPUT TAPE i, n, iterative list</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>READ TAPE i, LIST</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>READ TAPE i, iterative list</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>READ (i, n) list</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>READ (i, n) iterative list</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>READ (i) list</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>READ (i) iterative list</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>READ a, b, c,</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>STATEMENT</td>
<td>FORTRAN</td>
<td>I</td>
<td>II</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------</td>
<td>---</td>
<td>----</td>
</tr>
<tr>
<td>RECORD (i) a, b, c,</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RETURN</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REWIND i</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>STOP</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SUBROUTINE name (lk m, n, ...)</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TYPE n iterative list</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE OUTPUT TAPE i, n, list</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>WRITE OUTPUT TAPE i, n, iterative list</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>WRITE TAPE i, list</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>WRITE TAPE i, iterative list</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>WRITE (i, n) list</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>WRITE (i, n) iterative list</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>WRITE (i) list</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>WRITE (i) iterative list</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
3. EXPRESSION FORMATION

There are certain rules that will aid the programmer who does not work with Fortran constantly.

1. Variable operands must be previously defined.
   Explanation: This is accomplished by reading them or computing them prior to their use in an arithmetic statement.

2. Operators may not occupy adjacent positions.
   Explanation: They are separated by parentheses.
   \[ X = A \times (-20) \]

3. The expression must not be made up of mixed mode.
   Explanation: This can be either fixed or floating point but not both.
   \[ 20 \times Y \text{ creates a mixture of fixed and floating point.} \]
   \[ 20 \times Y \text{ is correct since both are floating point.} \]

4. A value may be assigned an exponent of different mode.
   Explanation: This is one exception to the mixed mode rule.

5. Spacing can be varied.
   Explanation: Items can be written together or spaced apart.

6. Operators may not be "assumed".
   Explanation: This is particularly true in multiplication.
   They must be written out.
   \[ 10 \times Y \]

4. HIERARCHY OF OPERATION

As with manual mathematics, the computer uses its own manner
and sequence.

1. Parentheses are considered first.

2. Exponentiation is carried out second.

   Explanation: If there are more than one operand to be raised to a power, parentheses must be used.

   Example:
   
   \[(gh)^10\] must be written \((g \times h)^{10}\)

3. Multiplication and division are carried out in their order of appearance from left to right.

   Example:
   
   \[A/3X\] is written \(A/(3 \times X)\)

4. Addition and subtraction are evaluated last, and occupy the same hierarchical level.

Parentheses

Parentheses are used to set off part of the arithmetic expression.

The enclosed part is evaluated first.

\[X = D (((C + B) / E^{1/2}))\]

The sum of C and B are calculated first. Their sum is divided by E and raised to the one-half power. D would then be added to the sum.

5. NESTED INPUT/OUTPUT STATEMENTS VERSUS THE DO LOOP EQUIVALENTS

Fortran I uses the DO loop, while Fortran II uses the nested iterative statements. The pattern set by Fortran I is that of making the innermost loop do the number of iterations. This continues with Fortran II by using the innermost nesting for this purpose. This compares to what we learned in data structures.
The do loops are easier to debug however, since the logic is easier to follow. A comparison can be made with the following read, print and punch routines:

```
READ 31, K, L, ((B(I,J), I = 1, K), J = 1, L)
READ 56, K, L
DO 25 J = 1, L
   DO 25 I = 1, K
   25 READ 58, A(I,J)
READ 32, A(I,J), Z(I,J), J = 1, 3, I = 1, 2
DO 40 I = 1, A
   DO 40 J = 1, 3
   40 READ 52, A(I,J), Z(I,J)
```

Figure FO-5
Nested Input Output Statements (con't.)

```
PRINT 25, (V(K, J), K = 1, 4), W(K), K = 1, 4
```

```
DO 10 K = 1, 4
  DO 20 J = 1, 4
    PRINT 25, W(K, J)
  END
  PRINT 25, H(K)
END
```

```
PUNCH 99, (NUM(I, J, K), K = 1, 2), I = 1, 20, 2
```

```
DO 200 I = 1, 20, 2
  DO 200 J = 1, 2
    DO 400 K = 1, 2
      PUNCH 99, NUM(I, J, K)
    END
  END
END
```

Figure F0-5 (Part-2)
6. FORTRAN SHORTCUTS

Space may be saved when a loop has within it an expression, whose variables do not change during the sequence through a loop. The space saving comes from evaluating the expression outside the loop, and retaining the result until it is needed.

\[
\text{DO } 60 \text{ I = I,N}
\]
\[
60 \text{ X(I) = YZ} \times \text{W(I)}/C
\]

An easier method:

\[
\text{HOLD I Y \times Z}/C
\]
\[
\text{DO } 60 \text{ I I,N}
\]
\[
60 \text{ X(I) = HOLD I \times W (I)}
\]

Repeated calculations can be improved by removing the redundant parts of the expressions.

\[
X (C \times D/A) \times \text{ COS(C \times D/A)}.
\]

These redundant expressions can be improved by using an area such as HOLD:

\[
\text{HOLDI = C \times D/A}
\]
\[
X = \text{HOLDI \times COS(HOLDI)}
\]

This makes it possible for the computer to make the computation of the expression one time only.

Polynomial Computation

The Fortran writing of:

\[
X = C + D \times Y + A \times Y^2 + E \times Y^3
\]

If we analyze the requirements for calculation, we find that there are two exponentiations, three multiplications and three additions.

The nested form of writing can be used to write the equation:
\[ X = C + Y*(D+Y*(A+Y*E)) \]

This method of presentation eliminates the multiplication and addition.

7. BUFFERING WITH FORTRAN

Input-output devices such as the teletype, card reader and punch work at speeds much slower than the computer. In the typical installation where computing time is quite costly, it is imperative that the computer be used for computing, with a minimal time for transmission to and from input-output devices.

Most computers now allow the computer to perform computations while data transmission is in process. The output data can be transmitted from memory to an intermediate buffer storage at high speed. This allows the computer to return to its computational tasks while data are being transmitted from the buffer to the output device, at the prescribed timing the output device requires. The process is similar for input data. This can be expanded for auxiliary storage such as disk and tape.

8. SYNTAX

Syntax is a study of language structure, not by a study of words themselves. A language syntax is a set of rules that dictate how the words, or basic elements, of the language are ordered to form meaningful phrases and statements.

Syntax for computer languages is not usually so simple as to have each character for one well-defined use letters representing themselves, symbols representing themselves, and symbols being used
their usual way. It is more usual that a single character is used in more than one way. For example, the asterisk is used for powers and also used to imply multiplication.

It is necessary that the algorithms can make decisions. To do so, we must make certain that each expression can be constructed one way only, according to syntactic rules.

9. ARRAY SEARCH

Search array Y, which has fifty elements, for its largest element. When the largest element is found, divide it into all the remaining elements of Y so that finally Y contains only elements with values less than or equal to one.

<table>
<thead>
<tr>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

```
do 30 i = 1, 50
    if (y(i) = y(kount)) goto 30, 50, 70
    kount = 1
70    continue
50    do 100 j = 1, 50
      y(j) = y(j)/x(kount)
```

Figure FO-9
10. SORT ASCENDING AND DESCENDING

A one dimensional array has forty elements. Prepare a program which will rank the array from smallest to largest, also write a program which will rank the array from largest to smallest.

1. Sort from largest to smallest:

```
DIMENSION Y(40)
DO 100 T = 1, 40
DO 100 J = T, 40
  IF(Y(J) .gt. Y(T)) THEN
    TEMP = Y(T)
    Y(T) = Y(J)
    Y(J) = TEMP
  END IF
100 CONTINUE
```

Figure F0-10
2. Sort from largest to smallest.

The outer DO loop, with one as index, ranks the elements one at a time (either largest to smallest or smallest to largest). Once an element is found, it is stored in Y (1). Then to get the next rank, the inner DO loop must examine Y (1+1) through Y (40). The inner DO loop searches the remaining elements each time to find the smallest (or largest).
11. INSTALLMENT NOTE PROGRAM

**Figure FO-11**
12. STATISTICS

Mean, Variance, Standard Deviation

Figure FO-12

Two Approaches to the Standard Deviation

Figure FO-12 (Part-2)
Figure SP-01
Figure SP-02
Statistics Program

Figure SP-03
Statistics Program

Figure SP-06
Statistics Program Print Out

<table>
<thead>
<tr>
<th>SCORES</th>
<th>NO. SCORES</th>
<th>MEAN SCORE</th>
<th>MEDIAN SCORE</th>
<th>VARIANCE</th>
<th>STANDARD DEVIATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIR CHARLES</td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHELLE</td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MERMAID</td>
<td>98</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFI</td>
<td>98</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUN</td>
<td>92</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIDO</td>
<td>92</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPOT</td>
<td>92</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUG</td>
<td>87</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOPPY</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READY</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TROUBLES</td>
<td>56</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCAN</td>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROMSER</td>
<td>52</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUNT</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIG BOY</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HAMILTON II</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REMON</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOWER</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAST BOY</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GIMMER</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Statistics Program Print Out Number 2

<table>
<thead>
<tr>
<th>SCORES</th>
<th>NO. SCORES</th>
<th>MEAN SCORE</th>
<th>MEDIAN SCORE</th>
<th>VARIANCE</th>
<th>STANDARD DEVIATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>13-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>13-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>15-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure SP-07
13. SELECTED FORTRAN PROGRAMS IN THE FORM OF ANSWERS TO QUESTIONS

Write the following Fortran programs:

13.1 Compound Interest Rate
13.2 Plotting
13.3 Integration by Simpson's Rule
13.4 Quadratic Equation
13.5 Linear Equation
13.6 Computed Go To
13.7 The Derivative
13.8 Evaluate a Polynomial
13.9 Functions
### 13.1 Compound Interest Rate

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>READ(5,20),BAL,XMON,RATE</td>
</tr>
<tr>
<td>3</td>
<td>WRITE(6,50),RATE</td>
</tr>
<tr>
<td>20</td>
<td>FORMAT:</td>
</tr>
<tr>
<td></td>
<td>XMON=</td>
</tr>
<tr>
<td></td>
<td>RATE = RATE/12.0</td>
</tr>
<tr>
<td></td>
<td>TOTAL = ((RATE%((1.0+RATE)*XMON))/((1.0+RATE)*XMON)-1.0)*BAL</td>
</tr>
<tr>
<td>DO</td>
<td>I = 1, NR</td>
</tr>
<tr>
<td></td>
<td>XINIT = BAL - RATE</td>
</tr>
<tr>
<td></td>
<td>FIN = TOTAL = XINIT</td>
</tr>
<tr>
<td></td>
<td>BAL = BAL - FIN</td>
</tr>
<tr>
<td></td>
<td>WRITE(6,70),I,XINIT,FIN,TOTAL,BAL</td>
</tr>
<tr>
<td>70</td>
<td>FORMAT:</td>
</tr>
<tr>
<td>90</td>
<td>CONTINUE</td>
</tr>
<tr>
<td></td>
<td>STOP</td>
</tr>
<tr>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>
13.2 PLOTTING

<table>
<thead>
<tr>
<th>FORTRAN STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIMENSION I(100), PLOT(1)</td>
</tr>
<tr>
<td>READ (1,11) I</td>
</tr>
<tr>
<td>WRITE</td>
</tr>
<tr>
<td>DO 20 J = 1, N</td>
</tr>
<tr>
<td>READ (1,11) IC(J)</td>
</tr>
<tr>
<td>WRITE(1,13)</td>
</tr>
<tr>
<td>IDOT = 10.52</td>
</tr>
<tr>
<td>IBLANK = 0.192</td>
</tr>
<tr>
<td>DO 50 J = 1, N</td>
</tr>
<tr>
<td>DO 50 K = 1, 12</td>
</tr>
<tr>
<td>IF (K - J) .EQ. 21, 22, 23</td>
</tr>
<tr>
<td>SPLOT(K) = IBLANK</td>
</tr>
<tr>
<td>GO TO 30</td>
</tr>
<tr>
<td>22 IF (K) .EQ. IDOT</td>
</tr>
<tr>
<td>GO TO 10</td>
</tr>
<tr>
<td>CONTINUE</td>
</tr>
<tr>
<td>WRITE</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>

Figure FO-13.2
13.3 INTEGRATION BY SIMPSON'S RULE

```
DIMENSION
*
5 READ (1,10)X,N
10 FORMAT
I = N/2
READ (1,20)(Y(J),J=1,N)
20 FORMAT
Y=0
DO 25 J = 2,N/2
25 Y = Y + Y(J)
Y0 = 0
DO 25 J = 2,N/2
25 YD = YD + Y(J)
R = Y(J) + Y(J+1)
S = R + YD + Y0 + Y0
AREA = S/3,XY
WRITE (1,30)AREA
30 FORMAT
PAUSE
GO TO 5
END
```

Figure FO-13.3
13.4 QUADRATIC EQUATION

```plaintext
5. READ (1, I0) A, B, C
10. FORMAT...
   X = B**2 - A + 4 * C
   Y = (-A - SQRT(X))/(2. * A)
   Z = (-A + SQRT(X))/(2. * A)
30. WRITE (1, 40) Y, Z
40. FORMAT
41. PAUSE.
50. GO TO 5
51. Y = -A/(2. * A)
   Z = Y
   GO TO 30
END.
```

Figure F0-13.4
13.5 LINEAR EQUATION

Figure FO-13.5
13.6 COMPUTED GO TO

<table>
<thead>
<tr>
<th>FORMATTED STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>10. READ (1,1) (x, y)</td>
</tr>
<tr>
<td>/</td>
</tr>
<tr>
<td>20. (T = 5\times x)</td>
</tr>
<tr>
<td>25. WRITE (1,2) (T)</td>
</tr>
<tr>
<td>26. FORMAT</td>
</tr>
<tr>
<td>PAUSE</td>
</tr>
<tr>
<td>30. GO TO 10</td>
</tr>
<tr>
<td>40. (T = 5\times y)</td>
</tr>
<tr>
<td>45. GO TO 20</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>

Figure F0-13.6
13.7 THE DERIVATIVE

```
DELIBL3 FOR X(t), Y(t)
10 READ (1,2) H, (X(t), Y(t), t = 1, T)
20 FORMAT
30 Z = (.75/H) * (Y(t) - Y(T)) - Y(T) - .15/H * (Y(t)- Y(A) ) +
40 WRITE (1,30) Z
30 FORMAT
PAUSE
60 GO TO 10
END
```

Figure F0-13.7
### 13.8 EVALUATE A POLYNOMIAL

The following is a program to evaluate a polynomial. The program takes the coefficients of the polynomial and an value of the variable and computes the polynomial value.

#### Program Code

```fortran
SUBROUTINE POLY(X, N, POLY)

DIMENSION A(50)

POLY = A(1)

DO 300 I = 2, N

300 POLY = POLY + X**I * A(I)

RETURN

END
```

---

**Figure FO-13.8**
13.9 FUNCTIONS

<table>
<thead>
<tr>
<th>FUNCTION MAX(T(90))</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIMENSION T(90)</td>
</tr>
<tr>
<td>MAX = T(1)</td>
</tr>
<tr>
<td>DO 90, I = 2, N</td>
</tr>
<tr>
<td>IF (MAX - T(I)) .GT. 0.90</td>
</tr>
<tr>
<td>GO TO 90, MAX = T(2)</td>
</tr>
<tr>
<td>GO CONTINUE</td>
</tr>
<tr>
<td>RETURN</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>

Figure FO-13.9
14. ARITHMETIC QUESTIONS AND ANSWERS

The answers immediately follow the questions for the first six numbers.

Examples

1. \( V = c + e^{(1./N)\times(R/2)-4} \)
   \[ V = c + e^{1/n \times r/2-4} \]

2. \( Y = (a + b)^{1/6} \)
   \[ Y = (A + B)^{1/6} \]

3. \( r = \frac{a(a + b)}{b^2 - c} \)
   \[ R = A*(A + B)/(B^2 - C) \]

4. \( V = \frac{4}{3} \pi r^3 \)
   \[ V = (4./3)\times3.14\times(R^3) \]

5. \( y = \frac{A + B}{C} \)
   \[ Y = (A + B)/C \]

6. \( x/y^{p-1} \)
   \[ X/Y^{p-1} \]
Questions

1. \((3 \cdot B)^{-1/3}\)
2. \((2 \cdot D \cdot C)^{(1/2)}\)
3. \((3 \cdot A \cdot B)^{(X-3)}\)
4. \(\frac{1}{8} \frac{(4 + A)}{3-B}\)
5. \(3 (A + B)\)
6. \(5A/2B\)
7. \(4A + 2\)
8. \(6(2/A)B\)
9. \((6 + 2/A) B\)
10. \(6 - 2/A - B\)
11. \((4B/2C)\)
12. \(((5-A)/(3-B))C\)
13. \(B/2+C\)
14. \(6 + A/2 - B\)
15. \(6 + A/3B\)
16. \(6 + 2/A\)
17. \(6 + 2*A\)
18. \(A - C + 4\)
19. \(y = \frac{a^2 + b}{b^2 - 2ab}\)
20. \(B*2**L-4. *P/C\)
21. \(3 \cdot \frac{\text{sales} \cdot \text{ocost}}{\text{Xinv} - \text{ucost}}^{1/2}\)
22. \(\frac{(y + a)/r}{f + 1} \cdot p^{1/3} + v \cdot 2\)
23. \[ h + \left(\frac{d}{e}\right)^2 \cdot \frac{f \cdot g \cdot r}{s + t} \]

24. \[ y - 3m^2 - \frac{r}{s^3} \]

Answers

1. \( (3A*B)^{-1/3} \)
2. \( (2DC)^{1/2} \)
3. \( (3AB)^{-3} \)
4. \( \frac{1}{8} * \left(\frac{4 + A}{2 - B}\right) \)
5. \( A + B \)
6. \( 2 + B \)
7. \( 4 \times A + 2 \)
8. \( \frac{2}{A*B} \)
9. \( (6 + 2/A) * B \)
10. \( 6 - 2/A - B \)
11. \( \frac{4*B}{((2*C)^P} \)
12. \( \frac{5 + A}{3 - B} \)
13. \( B/(2*C) \)
14. \( (6 + A)/(2 - B) \)
15. \( (6 + A)/(2*B) \)
16. \( A + 2/A \)
17. \( 6 + 2B \)
18. \( A - C + 4 \)
19. \( y - (A**2 + B)/(B**2 - 2*A*B) \)
20. \( b^2 - 4p/c \)
21. \( (3*\text{SALES} \times \text{OCOST})/(\text{XINV} - \text{UCOST})^{0.5} \)
22. \( ((Y + A)/4)/(F + 1) \times (P^{1.3} + (V*Z)/(B*W)) \)
23. \( (h + (D/E)^{2*3}((F*G*R)/(S + T))^{1.3}) \)
24. \( Y = 3*M*N^2 - R/S^2 \)

Activities

1. Write a Fortran program pertaining to the compound interest rate.
2. Write an installment note program.
4. Write a program and draw a flowchart for the quadratic equation.
5. Write a program to solve linear equations.
6. Prepare a program utilizing the computed GO TO statement.
7. Prepare a program to compute the derivative.
8. Write a program to evaluate a polynomial.
9. Write a subroutine subprogram to invert an array.

Use three arguments:

1. The given array
2. The array into which the inverted array will be placed.
3. An integer quantity that tells how long the arrays are. The array will be called TURBACK.

10. Write a function subprogram, which includes a dimension statement, to find the algebraically smallest quantity in
an array with a maximum of five hundred values. The function is to be called LEAST, and it will have two arguments:

1. The dummy array name.
2. An integer variable telling how many numbers are in the array.

Fortran Language Statements

Equivalence Statement

Form:

```
EQUIVALENCE (D, A, C(3))
```

1. The variables D, A, and C(3) will be stored in the same location in memory, however they can't be stored together at the same location at the same time.
2. This allows a saving of space, since after one variable is used, the next variable then occupies the space etc.

Subroutine Statement

Form:

```
SUBROUTINE NAME (A1, A2 A3...)
```

1. The Dimension statement is non-executable.
2. It tells the compiler the amount of storage to set aside for each variable.
3. Any number of variables, separated by commas, may be written in the statement.
Dimension Statement

Form:

DIMENSION B (40, 40)

1. The Dimension statement is non-executable.

Form:

READ 13 IMNAM, RATE

13 FORMAT F6.1, F6.3)

1. The format statement is non-executable.
2. It specifies input.
3. It specifies output.
4. The F specification transmits only floating point numbers to and from internal storage.

Go To Statement

Form:

GO TO (100, 101, 105, 120), L

1. Control is transferred to the statement numbers when the value of the indexing register is equal to each one.

Go To Statement (unconditional branch)

Form:

GO TO 15

An unconditional branch is made to statement 15.

Continue Statement

Form:

10 CONTINUE
X Specification Statement

Form:

```
READ 105, SRATE

105 FORMAT (33X F5.0)
```

1. It describes the number of positions to be ignored when a card is read.

2. Output causes blank characters to be inserted into the positions indicated. (There will be thirty-three blanks in this example.)

3. The numerical value of SRATE is punched into the thirty-fourth columns.

Read Statement

Form:

```
READ 110, RATE

110 FORMAT (E12.1)
```

1. It is nonexecutable.

2. It depicts the form in which the data will appear on input.

3. It shows how the data will be converted for storage in memory.

4. It also specifies, for output, the manner in which the data will be converted from memory and how it will appear in the output device for printing or punching.

5. The number is converted to internal floating point, (ENAM).
F Specification Statement

1. It tells the compiler the amount of storage to set aside for each variable.
2. Any number of variables, separated by commas, may be written in the statement.
3. The example presented here has two dimensions and will set aside sixteen hundred locations (forty times forty).

H Specification Statement

Form

READ 150
150 FORMAT (15H H GO TO TOWN)

1. The alphabetic information punched in the first fifteen columns of a card replace the fifteen characters that are in temporary storage. These fifteen characters in temporary storage were previously written in statement one hundred fifty.

I Specification Statement

Form:

READ 105, SRATE
105 FORMAT 33X F5.0

1. It describes the number of positions to be ignored when a card is read.
2. Output causes blank characters to be inserted into the positions.
3. It is the last statement of a DO loop, when the DO loop would cause a transfer type statement of the wrong kind.

**DO Statement**

**Form:**

```
DO 8 = 1, 10, 3
A(1) = 8
A(1,1) = 1,1
A(1,2) = 1,2
8 PRINT 9, A(1), A(1,1), A(1,2)
9 FORMAT (3 F8.0)
```

1. The DO statement tells the computer to execute repeatedly the statements which follow up to, and including the statement, with the statement number given (it is eight in this case).

2. The third number to the right of the equal sign is the amount of the increment for the index register.

3. After the statements have been executed ten times (n), control passes to the statement number given here as eight.

**If Statement**

**Form:**

```
IF(1NT-100) 20, 2, 4
```

The expression within the parentheses is evolved:

1. If the value is negative, control is transferred to the second control statement.

2. If the value is zero, control is transferred to the second
control statement.
3. If the value is positive, control is transferred to the third statement.

Arithmetic Statements

Form:

\[ b = d \]

1. The \( b \) is a variable, which can be subscripted.
2. The \( d \) is arithmetic in type, which is used to form a sequence of constants, operation symbols and variables.
3. The sequence connotes a series of calculations. The expression on the right hand side of the equal sign is completed, then the resulting numerical value is stored in the location assigned to the variable on the left side of the equal sign.

Call Statement

Form:

CALL TAX (BIC)

1. Tax is the name of a subroutine subprogram, and BIC is an expression of fixed or floating-point constants or variables.
2. The CALL statement calls the subroutines and does one of the following:
   a. It receives results back from the subprogram, or
   b. Sends data to the subprogram through an argument list.
Print Statement

Form:

PRINT 10 PROD, YTOT

10 FORMAT (F12.2, F10.2)

1. The two totals in the Format statement are printed.

Pause Statement

Form:

PAUSE 32

1. A halt is taken when the program reaches the pause statement.
2. Continue by pressing the start.

Slash and Repetition Statements

Form:

FORMAT (4/ E 10.2// (3F10.7))

1. The first slash causes the typewriter to start a new line.
2. Then skip a line.
3. Print R numbers per line until all numbers in the list are printed.

End Statement

Form:

END

1. It informs the compiler the translation is completed.

Type Statement

Form:
TYPE 100 TAX, ITEM

100 FORMAT (F10.0, F9.0)

1. The two variables are retrieved, converted and printed in the first nineteen positions.

Return Statement

Form:

1. Return indicates a subprogram has been completed and causes an unconditional branch back to the main program.

Stop Statement

Form:

STOP 999

1. It causes a halt.

Read Statement

Form:

READ 10, EARN

10 FORMAT (F10.2)

1. When READ is reached in the program a sufficient number of cards are read to satisfy the list of variables.

2. The format statement describes the data in detail.

3. The data is stored in memory as the floating point EARN.

Punch Statement

Form:

PUNCH 100, STAX, EARN

100 FORMAT (F10.0, 16)
1. This statement is similar to the read statement.

Function Statement

Form:

\[
\text{FUNCTION LOW (P, Q)}
\]

1. One argument must be included in the function statement.
2. The function name must consist of six alphabetic characters, and the first must be alphabetical.
3. The function statement operates when the name of the function is encountered in the arithmetic statement.
4. Listed arguments move data to the function routine, and a single quantity is returned to the calling program.
SECTION IV

RPG
IV  RPG

The purpose of this section is to provide models and information which will aid in the preparation for the DPMA test. The Basic Assembler program presented here is used as a comparison for the invoicing program written in Report Program Generator (RPG).

The reader is advised to review all the pertinent forms for RPG since they are considered important for the test. The latter part of the section provides the necessary information to perform the Swanson Type programs with RPG, and presents the assignment that could make this text last a year or longer.
Object Program Logic

Figure RPG-0
PRE-BILLING CALCULATION WITH INVENTORY CONTROL

This example illustrates one of numerous approaches to an order-processing/inventory control job. The application has been arbitrarily slanted to a distribution business (perhaps a mail-order house) with customer orders to be filled from warehouse stock. An attempt has been made to be reasonably realistic in the application, including the complexities of such a multipurpose operation.

BASIC ASSUMPTIONS

1. A card has been keypunched:
   a. For each item line on a customer order—Card 9, no X in col. 11
   b. For each item line on a customer return—Card, X in col. 11
   c. For each item line on a stock receipt (or purchase-order cards are used as stock receipt cards) — Card 5
   d. For each stock adjustment — Card 6: No X in col. 11 to reduce on hand, X in col. 11
   e. For each item on a stock purchase order — Card 7, no X in col 11 when ordered, X in col. 11 if order is cancelled or reduced
   f. For a new stock item or a change in price, description, warehouse location, etc. (Obsolete master cards are removed manually or, at least, separately from this operation.)

2. An Inventory Master Card file exists, with one card per item carried in stock. Changes to the file are made manually, or in some other data processing operation (i.e., addition and deletion of items, changes in price, warehouse location, etc).
3. It is desired to process customer order-item cards against inventory records before attempting to fill the orders in the warehouse. At the same time, the inventory records will be updated and an up-to-date inventory report prepared. The customer-order cards are thereafter ready for invoicing. (The cards could be sorted by warehouse location prior to invoicing.) A copy of the invoice, or the cards themselves, serve as order-picking medium, i.e., either sequential or bulk picking is employed. If orders are processed once daily on this basis, the inventory records are always up to date.

4. If the quantity in-hand is insufficient to satisfy the quantity in the customer-order card, no partial quantity will be applied for that item. The item order:
   a. Will be marked "cancelled" if no stock is on order; or
   b. Will be marked for back order if not previously back-ordered, and provided stock is on order; or
   c. Will be marked "cancelled" if previously back-ordered.

5. Where previously back-ordered item cards are re-entered, they are to receive priority for available inventory.

6. Some items have a lower unit selling price when at least the specified criterion quantity is ordered by the customer.

7. Stock adjustments are made without attempt at modifying the unit cost of the item.

8. a. Besides price extension, gross profit is to be included in the item detail cards for a subsequent report by merchandise class and division, and by Stock No. (The first digit of Stock No. represents merchandising division, the second the
classification within division.)

b. Value of inventory on hand (average cost basis) is to be continually available.

c. Available quantity (on-hand plus on-order) less than an established minimum to be signalled.

PROCEDURAL DETAILS

1. Safeguards

   a. Certain control totals will be carried, partially as audit trails. Control totals are presumed to have been established for the various kinds of transaction cards, so that new on-order totals can be proved out.

   b. Customer-order detail cards that are being cancelled will be identified. If such a card is re-entered, it is selected out, and calculation for it is bypassed.

   c. Matched old master cards (for which new ones are created) will be identified, and selected to a separate stacker. If such a card is accidentally re-entered, the entire stock-number group is selected to separate stacker, and calculation is bypassed.

   d. The entire stock-number group (except the first card) is selected to a separate stacker, processing is bypassed, and the system halts after the second card, whenever there is more than one master card for a group.

   e. The entire stock-number group is selected, and calculations are bypassed, when a master card with a negative on-hand quantity has been read. When a negative on-hand quantity
is created as a result of calculation, the cards from the point of error are selected, and calculations are bypassed.

f. Whenever the blank trailer card is missing or mispositioned within the group, all cards in the group from the point of the error detection are selected, the system halts, and further calculation is bypassed for the group.

g. Unmatched transaction cards, including the trailing blank card, are selected, and calculations are bypassed.

h. If the on-order quantity turns negative, the system halts. The inventory report also indicates this condition.

i. For known error conditions that affect the new inventory values, the data is omitted.

2. Any merchandise receipts, stock adjustments, and customer returns precede order-item details, so that the customer orders are correctly applied to the latest on-hand status. Stock purchase-order cards are also placed ahead of customer order details, because it was decided not to back-order items for which no stock is on order. Former back-order cards precede other order-item cards to get first chance at on-hand goods.

3. The cards are assumed to be in ascending sequence by Stock No. Inventory master cards are to be in the primary feed of the MFCM—preceded by a single card to read in today's date. All other cards will be placed in the secondary feed.

A previous operation has placed a blank card at the end of each Stock No. group of secondary file cards. These blank cards will become the new (updated) inventory master cards for stock numbers for which
there are transactions. (These blank cards were merged in on the MFCM of the Model 20, or they could have been merged on a collator.)

4. Stacker Selection

a. The date header card is directed to stacker 1; any other stacker would do equally well.

b. All old inventory master cards with stock numbers, for which there are transaction cards in the secondary file, are directed to stacker 1 (the normal stacker—chosen to contain obsoleted cards), because a new inventory master card will be punched and placed in stacker 2.

Each unmatched old inventory master is selected to stacker 2, because no new master is punched in such case.

Stacker 2 ultimately contains the complete up-to-date inventory master file (except for known error-condition cards) consisting of new cards where transactions occurred and old masters where no transactions applied.

c. Stacker 3, receives the customer order-item cards, ready for warehouse picking (if cancelled and BO (back-orders) are sorted out), or to be sorted on order and account numbers for invoicing.

d. Stacker 4 has been assigned to unmatched transaction cards (secondary file), and to all other detected error-condition cards.

e. Stacker 5 has been assigned to stock orders, receipts, adjustments, and merchandise returns. These may also be left together with the other transaction cards by directing them
to stacker 3 instead; they could easily be segregated by sorting cols. 1 and 11.

Card Layouts

![Image of Multiple-Card Layout Form](image-url)
Study of card layouts and the systems flow as seen in Figure P-1 will clarify the details of the operation. The report has been laid out to fit within the 120-position print span of all IBM 2203 and 1403 Printers attachable to model 20. Explanation of specifications sheets follows.
FILE DESCRIPTION SPECIFICATIONS (Figure P-1)

The file inventory master cards is named OLDMASTHR, and associated with the primary hopper of the MFCM. It is defined as a combined file (C in col. 15) so that stacker selection may be performed via output specifications, and to allow punching of a code for "obsolete" at output time into those old masters that are replaced as a result of new transactions.

The detail transaction cards are assigned to the file named TRSACTN, and associated with the secondary hopper of the MFCM. Stacker selection is dependent on calculation operations; therefore—and because output is required to some customer-order item cards—TRSACTN is a combined file.

The input files are in ascending sequence (A in col. 18). A
sequence is required, and must be uniform for the input files, when
matching of records is called for in two or more files. If col. 17 is
blank, or contains E, for all input files, the LR indicator does not
turn on until all input files are exhausted.

The printer is associated with an output file named REPORT.

File Description Specifications

Figure P-1

INPUT SPECIFICATIONS (Figure P-2)

Because the file OLDMASTR is specified ahead of the TRSACTN
file, it is therefore the primary file; i.e., matching cards from the
OLDMASTR file are processed ahead of their matching TRSACTN file cards.
**Figure P-2**

**Input Specifications**

**Figure P-3**
INVENTORY MASTER CARDS—OLDMASTR File (Page 02)

The old Inventory Master cards are identified by 0 in col. 1, and assigned indicator 01. Since they are the only card type in the file, apart from the initial single Date card, an alphabetic code is specified in Sequence (cols. 15–16). If any other undefined card types (besides the Date or Master card) appear in the file, the system halts.

Lines 02–18 contain the normal specifications for reading those fields from the old Inventory Master cards that may be needed for processing of the application. Fields defined as numeric are used in calculations, edit operations, or numeric compare. Points of special interest are:

1. Stock No. is defined as numeric to allow formatting in the output by edit word, and to simplify detection of an obsolete master card (see 4 below).

2. The files are matched and sequence checked on Stock No. (M1 in cols. 61–62 for Stock No.).

3. The Ll indicator is turned on for the first card of each stock-number group (Ll in Control Level for Stock No.). Ll is not used in this program for total printing or punching, it is used solely to inherent connection with matching of files, and Ll is not needed merely because Ml is assigned.

4. Whenever an old Master Card is replaced by a new one, to reflect transactions, the old card is overpunched with an 11-punch in col. 7 at output time (Fig. P7, line 06), to mark it as obsolete. If such a card is accidentally re-entered next time, indicator 97 on the 11-punch causes the
Stock No. to read in as negative (a matching-field sequence error does not, however, arise because all zone punches are eliminated from the matching-fields operations of a numeric field).

5. Indicator 99 turns on if the Quantity On-Hand is negative in the old Inventory Master card. Such a card should never appear, because subsequent specifications (Fig. P4) cause output to be bypassed if On-Hand turns negative.

6. Indicator 20 turns on if the Criterion Quantity field is zero. The zero code indicates that only Unit Price A applies, and that the Price-B field is to remain blank both in the report and in a new Inventory Master card.

7. Col. 47 appears twice among the input fields—the first time as part of a normal numeric field: the second time with another name and as a single-column alphameric field. If col. 47 is X-punched (11 -punch), Quantity Sold Last Year does not apply because the item is new this year. The word NEW is then to appear in the report, and the field is only to contain an X-punch in a new Inventory Master card. But a numeric field that is blank or zero with an X-overpunch in the units position will set on an indicator for Zero or Blank—not for Minus. Therefore, the column that contains the X-punch for "new" is separately defined as alphameric. It can then be tested for a minus done by a TESTZ calculation specification.

8. Stacker assignment is not known until calculations are per-
formed. It must therefore be specified at output time.

DATE CARD OLDMASTER FILE (Fig. P3)

The single Date card at the front of the file is identified by an X-punch in col. 1 and assigned indicator 09. The date is stored in a field given the name DATE. It is defined as numeric to allow editing.

No matching is specified for this card. It is therefore processed first. The date card is to enter the normal stacker for the MFOM primary hopper and therefore need not have stacker selection specified. However, when no output operation is to be performed on a combined-file card type, and the desired stacker number is known at input time, a stacker-selection specification, even for the normal stacker, should be given in the input specifications; this maximized I/C overlap. (For a single card in an entire file, this is of course insignificant). The File Name need not be repeated where no others intervened.

TRANSACTION CARDS (Except Blank Trailer) - TRSACTN FILE (Fig. p4)

The four types are identified and assigned separate indicators. The customer-order or merchandise-return item card is checked for digit rather than character 9 because back orders have an X-overpunch in col. 1.

Stacker selection is dependent on calculations, and is therefore assigned in the output specifications. In the case of card type 9 (indicator 15), output to the card is also required; this precludes stacker selection in the input specifications.
Points to note:

1. Indicator 21 is turned on for order-item cards that were previously back-ordered: 11/9 in the low-order, or sole, position of a numeric field indicates a negative value. (Back-order cards are so designated at output time—Fig. P7, line 17.) The field BOCARD is not used in the program; it is assigned only so that a Field Indicator may be set. Alternatively, a separate card-type Resulting Indicator could have been assigned via an OR line.

2. The same name is assigned to Stock No. here as for the OLDMA¥TR file, to conserve core storage space. No harm is done because there is no situation in this program where the distinction needs to be preserved.

3. When an order-item cannot be filled, and is not to be back-ordered, col. 7 of the card is overpunched with an 11-punch (Fig. P7, line 18) to designate "cancelled." If such a card is inadvertently reentered, indicator 98 turns on because the 11-overpunch causes Stock No. to be read as negative.

4. Indicator 22 distinguishes between order-item and merchandise-return cards—both card-type Resulting Indicator 15.

5. The field UNCOST applies only to Receipt cards. No harm is done reading it also from card types with Resulting Indicators 12, 13, and 15, because utilization in the calculation specifications is confined to card type 11.
(Fig. P5, line 06). If it were necessary to restrict the
input of this field to Receipt cards, the indicator
number (11) would be entered in Field-Record Relation
(cols. 63-64).

BLANK TRAILER CARD—TRSACTN FILE (Fig. P3)

The trailer cards—destined to become new Inventory Master cards—
are identified by absence of a punch in col. 1, and are assigned
indicator 19.

The blank trailer card at the end of each stock-number group
in the TRSACTN file is not matched (no entry in Matching Fields)
against the OLDMASTR file; therefore, it is processed immediately
after the card it follows in the same file, before the Inventory Master
card for the next Stock No.

CALCULATION SPECIFICATIONS (Fig. R4)

In order to minimize the need for conditioning indicators
(Indicators, cols. 9-17), branching (GOTO) over entire sections has
been employed to bypass a series of inapplicable calculation specifi-
cations.

Where practical, specifications lines are discussed sequentially.
In some areas, however, it is preferable, for clarity, to relate non-
consecutive lines. Note: In several instances, result fields are
defined as smaller than the theoretically possible maximum. We assumed
that knowledge of the particular business indicated that these field
sizes are adequate for the actual figures that could occur.
Calculations Specifications

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Digits</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>SETON</td>
<td>GOTO END</td>
<td>9.3</td>
<td>IDENT_NEXT_CARD</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.9</td>
<td>GOTO END</td>
<td></td>
<td>DATE_RVPAS_CRL</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>SETOFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>SETON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>SETOFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>SETOFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>SETOFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>SETOFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>MOV STRAND OLCNO</td>
<td>4.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>SETON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>SETON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>SETON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>SETON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>GOTO END</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>SETON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>GOTO END</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>SETON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: This completes general checks for principal error conditions.

Figure P4

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Factor 1</th>
<th>Operation</th>
<th>Factor 2</th>
<th>Result Field</th>
<th>Digits</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>GOTO MASTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>HREF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>GOTO MASTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>MSTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure P5
Calculations Specifications (con't.)

Date Card (Card-Type Resulting Indicator 09)  
(Fig. P4, lines 01 and 02)

No calculation operations are performed on this card. Indicator 93 is turned on (line 01) solely for use in a subsequent check on proper card-type sequence (line 05). The entries in line 02 cause branching to the end of the calculation specifications (page 06, line 20), so that No. 9 need not be specified in Indicators in subsequent lines.

Error Control - (Fig. P4, lines 03-18)

Calculation specifications are employed to test for certain error conditions. Where an error is recognized that affects only the
individual card, calculations are bypassed for that card, and the card will be selected by output specifications to stacker 4. Where the effect pervades the entire stock-number group, all calculations for the group are bypassed from the point of error recognition, and those cards will be selected to stacker 4. For certain error situations, the system is also halted.

Indicator 90 is set on for all of the major error conditions tested for, and is used to specify the bypassing of calculations and the selection (see output specifications) of the group to stacker 4. Specifications line 03 clears indicator 90 at the beginning of each control group (i.e., stock-number group), so that the error actions do not carry through to the next group.

Indicator 91 is turned on in line 17 if indicator 19 (blank card) is on. Next program cycle, indicators 90 and H2 are turned on if indicator 91 is still on when the instructions in line 14 are reached by the program. However, if indicator L1 (first card of control group) is on when the instructions in line 07 are reached, indicator 91 is turned off.

Thus, an error is signalled (90 and H2 are turned on) if there is no control break (L1) following a blank card (91 turned on by 19). Trailer card present but not at end of group.

Duplicate Master or Sequence Step-Down

In line 08, the stock number in the old Inventory Master card is compared algebraically with that of the previous old Master card. If the number is the same (duplicate master) or lower, H1 is turned on to halt the system after the card has been processed. In line 09,
the next master card is processed.

In line 10, indicator 90 is turned on if H1 was turned on in line 08, so that all processing for the remainder of the group will be bypassed, and the cards selected to stacker 4.

Note: Because the matching fields assigned in the input specifications were defined as numeric (line 03 of page 02, and line 08 of Fig. P3), the sequence check performed as a result of the M1 specification ignores sign. For that reason, the H1 indicator is also turned on for a negative comparison result—otherwise a duplicate is not detected if one card is positive and one negative in the stock-number field. However, indicators 97 and 98 also signal a negative stock number, but without a halt.

Obsolete Old Inventory Master Card

Indicator 97 turns on if the Stock No. in the old Master card is negative, signalling reentry into the operation of a previously obsoleted card. In line 13, indicator 90 is set on for that condition.

Negative on-Hand in Old Inventory Master Card

Indicator 99 turns on if the On-Hand field is negative at input time of the old Master card. In line 11, indicator 90 is set on for that condition.

Cancelled Order-Item Card

Indicator 98 turns on when a transaction card with a negative stock-number field is read. This signals reentry of a previously cancelled order-item card.
Indicator 98 is used to specify bypassing of calculations for that card only (see line 15), and its selection to stacker 4, but the remainder of the group is processed normally because it is not otherwise affected.

Unmatched Transaction Cards

The specifications in line 12 cause indicator 90 to be turned on for unmatched cards (NMR), other than Inventory Master cards (N01), and other than blank trailer cards (N19) which are always unmatched.

Bypassing Calculations for the Error Group

In line 18 the program branches to END (line 20 on Fig. P6) when indicator 90 is on. This makes detail output the next operation, omitting all calculations below line 17 on Fig. P4.

Line 19 illustrates use of a Comments card (* in col. 7). It will be printed during generation as punched, but otherwise it does not enter the generation process. (It is checked for proper position, based on cols. 1-6.)

Bypassing Detail-Card Operations on Master Cards
(Fig. P4, line 20 and Fig. P5, lines 01-03)

Line 20 of page 04 provides program skipping past all the specifications lines that do not apply to the new Inventory Master card (i.e., the blank trailer card). This minimizes the need for N19 specifications in Indicators in subsequent lines.

In line 01 of page 05 the average Unit Cost from the old Inventory Master card is saved for later determination of cost trend when compared with new merchandise costs.
In line 02, all calculations are terminated for old Inventory Master cards that will be replaced by new ones (i.e., there are matching transaction cards).

In line 03, the program skips—for old Master cards that are to be retained (i.e., there are no transactions)—to the same point at which calculations are resumed for new Inventory Master cards. This permits uniform preparation of report data for both situations.

Merchandise-Receipt, Stock-Adjustment, and Stock-Order Cards (lines 04–11 of Fig. P5)

In line 04, the On-Order quantity is revised to reflect Merchandise Receipts, new Purchase Stock Orders, and cancellation of Stock Orders. Cards 5 and 7 are so coded in col. 11 that addition provides the proper algebraic operation (see Figure P00). The system is halted if the operation results in a negative On-Order quantity. (Indicator 90 is not turned on, because such an error was not deemed of sufficient significance to require bypassing of the remainder of the group.)

Line 05 provides for extending the cost of a stock adjustment, based on last-known unit cost, so that the value of the inventory may be adjusted (in line 07). A new work field (CSTEST) is set up for the product.

Line 06 provides for the same operation as line 05, but using the specific unit of cost at which new merchandise was received.

In line 07, the extended cost of an Adjustment or Merchandise Receipt is algebraically subtracted from the total Inventory Value of the stock item. The signs in cards 5 and 6 are appropriately coded.
In line 08, the On-Hand quantity is updated to reflect Receipts and Adjustments. Indicator 90 is turned on if On-Hand has become negative; further calculations are then by-passed for that stock-number group (by entry in line 09), and the cards from this point on are selected to stacker 4 (output specifications).

In line 10, a new Average Unit Cost is established during processing of Receipt cards, because each of these cards contain unit cost. (In lines 06 and 07 we adjusted the Inventory Value to reflect the cost of the new Receipt proportionately.) The quotient is half-adjusted.

Division by zero is not permitted, nor meaningful. Indicator 26 (turned on in line 08 if On-Hand was greater than zero) is therefore a conditioning indicator.

Line 11 causes termination of calculations for cards 5, 6 and 7. Order-Item and Merchandise-Return cards cannot cause On-Hand to turn negative. If On-Hand was already negative, entries in lines 08 and 09 caused branching to END. Therefore, indicator 23 turns only for a customer order-item card containing a quantity larger than the positive or zero On-Hand quantity.

Lines 13-15 are executed only to handle the insufficient-stock situation (i.e., indicator 23 is on). In accordance with our Basic Assumptions:

1. No order-item will be partially filled.

2. No item card will be back-ordered if it was previously back-ordered.
3. No item will be back-ordered unless merchandise is on order.

In line 13, the quantity is added back to On-Hand to restore the prior status.

In line 14, indicator 24 is turned on if Quantity On-Order is greater than zero (COMP operation), provided the card was not previously back-ordered (N21—see Fig. P3, line 07). Indicators 23 and 24 determine, in the output specifications, whether the card is to be identified as Back-Ordered or Cancelled (Fig. P7, lines 17 and 18).

Indicator 24 is turned off each cycle (see Fig. P4, line 06) before this point is reached, because line 14 is not executed each time. Incorrect card identification in col. 1 would otherwise be punched when non-back-order cards follow a back-order card.

Line 15 causes branching to the end of the calculation specifications for order-item cards that could not be filled. The specifications in lines 02-10 of Fig. P6 will not be executed for these cases.

On Fig. P6, lines 02 and 03, respectively, set on indicator 27 if the customer-order or merchandise-return quantity is equal to or greater than the Criterion Quantity that qualifies for Price B.

We are only interested in the Resulting Indicator, not the actual result quantity. However, an arithmetic operation requires a result field. In order not to waste core storage space, a field only temporarily needed elsewhere (Fig. P5), but now available, has been utilized. A numeric Compare operation is always algebraic, therefore a more complex routine would have had to be substituted for the ADD operation in line 03 (where QTY is negative) if COMP were to be used instead.
Line 04 places Price into a new field, UNPRIC, which will be used for the unit-price factor in the selling-price extension.

In line 05, Price B is substituted for Price A in the UNPRIC field—but only provided the quantity in the Order-Item or Merchandise-Return card satisfied the criterion (lines 02 and 03) and provided Criterion Quantity was not ) (N20—see Fig. P2, line 06): zero in col. 22 indicates that Price A applies in all cases.

In line 06, the quantity in the item card is multiplied by the unit price previously selected (lines 02-05). The new field, EXTPRI, will be negative for a Merchandise-Return card, because quantity is negative.

In line 07, cost of the item sale or return is calculated, using the Average Unit Cost as updated during processing of any stock Receipt cards (Fig. P5, line 10). Again, the same work field CSTEXT, is utilized, because the product is not needed beyond line 08.

In line 08, gross profit is calculated for each item card. For merchandise returns, the sign is automatically reversed: -EXTPRI-(-) CSTEXT = -GRSPRO (unless selling price is less than Average Unit Cost).

In line 09, Quantity Sold This Year is updated for this item card. Returns reduce the value, because quantity in these cards is negative. Because it is possible for returns early in a year to exceed sales, provision is made for a negative total (Fig. P11, line 17, edit word).

Line 10 terminates calculations for card 9.
New Inventory Totals (lines 11-19, Fig. P6)

This section contains the specifications for completing the data needed (1) to punch the new inventory Master cards for stock numbers with transactions, and (2) to print the Inventory Status Report for all stock numbers.

No conditioning indicators are required because the program has been instructed, in earlier lines, to branch past this section—to END (line 20)—for all card types except blank trailer cards or unmatched (i.e., no transaction) old Master cards.

Output Specifications

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Constant or Edit Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>OLD MASTER</td>
</tr>
<tr>
<td>01</td>
<td>OR</td>
</tr>
<tr>
<td>02</td>
<td>OR</td>
</tr>
<tr>
<td>03</td>
<td>0</td>
</tr>
<tr>
<td>04</td>
<td>MANDO</td>
</tr>
<tr>
<td>05</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Figure P7
Pre-Billing - With Inventory Control

![Figure P8](image)

Output Specifications

![Figure P9](image)
Pre-Billing with Inventory Control

![Figure P10](image)

Output Specifications

![Figure P11](image)
Pre-Billing with Inventory Control

Line 16 is not needed when there are no transactions; but there is no harm in executing it. Although there is no change in Average Unit Cost when there are no Merchandise-Receipt cards in the group, line 15 (in conjunction with line 01, Fig. P5) provides a uniform method of determining cost trend that sets the indicators appropriately, regardless of whether there has been a Receipt.

Line 11 is the destination point to which the program branched from page 04, line 20 (blank trailer card) or page 05, line 03 (unmatched old Inventory Master card).

Line 12 provides for determining whether the item is new this year (X-punch in col. 47 of old Master card—see line 11, Fig. P2).

Indicator 30 will be used in output specifications to control punching into cols. 43-47, and printing in print positions 51-59.
In line 13, the available quantity (On-Hand + On-Order) is calculated for the report.

In line 14, indicator 31 is turned on if the available quantity is less than the minimum specified in the old Master card, so that this condition can be signalled by a symbol in the report (print position 120).

In line 16, the updated Inventory Value is calculated after all transactions have been processed.

Lines 17-19 contain the specifications for summing quantity On-Hand, Quantity On-Order, and Inventory Value for report grand totals.

The first two serve only as audit trails and control totals, to balance out former totals with control totals for Receipts, Adjustments, Stock Orders, Merchandise Returns, Back-Orders, and Order-Item cards. The Inventory Value total is also important as far as a management figure.

Line 20 represents merely the destination point to which the program branched from a number of previous lines when calculations were complete. It is followed by detail-time output.

Output Format Specifications (Fig. P7)

All output is at detail time (D or H in col. 15) except for grand totals, based on LR indicator which terminates the job after total-output time.

Old-Inventory Master Cards—Oldmastr File (Fig. P7, lines 01-06)

Lines 01-03 specify different stacker selection for card types
in an OR relationship. Cards with major errors (indicator 90 - see Fig. P4) are selected to stacker 4; the remainder (i.e., the bulk) are selected to stacker 2 if unmatched (NMR), or stacker 1 if matched (MR). Stacker 1—the normal stacker—need not be specified.

Thus, when a new master card will be created because there were transactions, the matched old Master is directed to pocket 1; if no new Master is created, it is directed to stacker 2, which will also receive the newly punched Masters for groups with transactions.

Indicator 01 is needed:

1. To prevent old Master cards of the following stock-number groups being passed through output operations, without being read, in the program cycles during which matched secondary cards are being processed (MR on).

2. To prevent an old Master card being passed through output operations, without being read, during the detail-time output preceding the reading of the first card at lp time (MR is then off; thus, NMR would apply).

3. To prevent performance of this output for the Date card (during whose processing NMR applies). The Date card was specified as requiring input only, by the stacker selection designated for it in the input specifications.

Line 04 specifies that obsolete old Inventory Master cards (which are replaced by the trailer card of the matched transaction-card group) are to receive an 11-punch in col. 7. This is the safeguard against accidental reentry of these cards next time (see indicator 97: Fig. P2, line 02 and Fig. P4, line 13).
Note: Indicators in File-Identification lines of card types in an OR relationship are tested in sequence. If indicator 90 is on, line 01 is applied and therefore, N90 is not needed in the next two lines. However, in line 04, N90 is necessary, because each Field-Description is considered separately for all card types in an OR relationship.

TRANSACTION CARDS: RECEIPTS, ADJUSTMENTS, STOCK ORDERS, AND ERRORS - TRSACTN FILE (Fig. P7, Lines 07-12)

Cards of groups with major recognized errors are selected to stacker 4. A previously cancelled order-item card that was inadvertently reentered (indicator 98--see page 03, line 08) is also selected to stacker 4. Fifteen is specified in line 08 (with indicator 98) so that additional cards following a cancelled order-item card are not also selected. Indicator 98, once on, is not reset until the next transaction card other than a blank is read.

Receipt, Stock-Adjustment, and Stock-Order cards are selected to stacker 5, they could instead be directed to stacker 3 with the order-item cards and subsequently sorted apart on card No. (col. 1).

ORDER-ITEM AND MERCHANDISE-RETURN CARDS-TRSACTN FILE (Fig. P7, Lines 13-19 and Fig. P8, Lines 01-09)

By the entries in lines 13-16, Merchandise-Return cards (indicator 22 on--see Fig. P3, line 09) are directed to stacker 5, whereas order-item cards are selected to stacker 3. The Returns cards could also, of course, be selected to stacker 3, and subsequently sorted apart by the X-overpunch in col. 11. The file name need not be repeated in line 13.
Line 17 provides for an 11-overpunch in col. 1 for order-item cards being back-ordered. (See Fig. P5, lines 12 and 14, for indicators.)

Line 18 specifies an 11-overpunch in col. 7 for order-items to be cancelled. (See page 05, lines 12 and 14, for indicators.)

Line 19 on page 07 and lines 01-05 on page 08 provide for punching of the pertinent data. Descriptions (line 19) are not punched the first time these cards are processed. The other fields (lines 01-05) are not punched into cards now being back-ordered or cancelled (indicator 23 on); these fields will be punched into the back-ordered cards when they are reprocessed, if the order-item is then filled.

Lines 06-09 provide for document printing (interpreting) on the Order-Item and Merchandise-Return cards, on an MFCM Model Al.

Warehouse location (line 08) is printed only if the item was filled, because the goods could be at a different location when new merchandise is received and the back-orders are filled.

The other three items are interpreted the first time the card is processed (to facilitate card handling), and are therefore not printed again on previously back-ordered cards.

Stock No., Quantity, and Warehouse Location are printed by print head 1; Account No. is printed by print head 2.

Stock No. (line 06) is edited with hyphens between digit positions two and three, and between the fifth and sixth (the presumed self-check digit). The third hyphen in the edit word is the status portion and identifies a cancelled card. All leading zeros, except the first, are preserved.
Zero Suppress is used to eliminate leading zeros in Account No. (line 09).

Note: These cards hereafter contain all the information needed to:

1. Run invoices.
2. Serve as warehouse picking tickets.
3. Run sales, cost-of-sales, and gross profit reports by stock number and merchandise class.

PUNCHING NEW MASTER CARDS (BLANK TRAILER CARDS) -- TRSACTN FILE (Fig. P8, Lines 10-19 and Fig. P9, Lines 01-11)

The pertinent fixed data from the old Master card and the updated variable information are specified for punching as per the card layout (Fig. PO).

If Criterion Quantity was 0 (indicator 20 on--see Fig. P2, line 06), the field for Price (line 15) is left blank. If the item is new this year (indicator 30 is on--see Fig. P6, line 12), the single-position alphanemic NEWITM field (consisting of an 11) is punched into col. 47, line 02. If the item existed last year, the five-digit numeric field LASTYR is punched into cols. 43-47 (line 01). If cost trend is up (indicator 32 is on), a plus (12/6/8) is punched into col. 75 (line 09); if it is down (indicator 33 is on), a minus (11) is punched (line 10, fig. P9); if there was no change in merchandise cost (indicators 32 and 33 are off), col. 75 is left
however, 10 is only on at detail time; therefore, detail output time is the simplest way to handle the operation.)

The heading consists of constants, with one exception, the output field PAGE is specified. This is the only field name (as contrasted with constants) that can provide other than blank or zeros before the first card has been read (i.e., when LP is on). Page No. 1 will be printed in the first heading line of the first page (it is not possible to start with any other value before a card has been read); it will be incremented by 1 before printing on the first line of each succeeding page. Zero Suppress is specified to eliminate leading zeros.

Lines 08-13 contain the specifications for the first print line of column headings, 3 lines beneath the report heading. The form is single-spaced after printing.

The column headings, too, are to appear on each page (first and overflow pages). The file name need not be repeated.

Lines 14-20 contain the specifications for the second print-line of column headings. A single space follows printing.

Lines 01-06 on Fig. P11 take care of the third print line of column headings. After printing, the form is advanced to the next channel -2 punch.

PRINTING THE ITEM LINES-REPORT (Fig. P11, Lines 07-18 and Fig. P12, Lines 01-10)

Lines 07 and 08 specify that the data is to be printed at detail-time (D in col. 15) while processing either:

1. A former blank trailer card (indicator 19 on) that does
not belong to a recognized error group (N90—see Fig. P4; and Fig. 05, line 08); or

2. An unmatched (NMR) old Inventory Master card (indicator 01 on) that does not belong to a recognized error group (N90).

Thus one line will be printed per stock number, showing the original old Inventory Master card data for items without new transactions (NMR), and the updated information where transaction cards exist.

Points to note:

1. In lines 11, 12, 13, 15, and 17 on page 11, the edit word is designed so that one 0 is printed when the quantity is completely zero, and a minus sign is printed for negative values in fields that can be negative. In lines 01, 02, 04 and 07 on Fig. P12, the edit word provides for printing of .00 when the amount field is all-zero. The edit word in line 07 of Fig. P12 also provides for a floating dollar sign.

2. The maximum number of leading zeros (i.e., all but one) is preserved for the Stock No., in line 18 on page 11, and hyphens separate merchandise class from the remainder of the number, and the principal number from the self-check digit.

3. The dates-lines 09 and 091 or Fig. P12 are edited to be printed with slashes between Month, Day, and Year. There is no point in placing a 0 in the edit word; the date can at most have one leading zero (months 01–09), and its suppression cannot be prevented by an edit-word entry.

4. Line 091 on Fig. P12 illustrates insertion of a specifications
line that had been forgotten initially, by assigning it a line number sequentially between two pre-printed numbers.

5. Lines 15 and 16 on page 11 cause the Quantity Sold Last Year to be printed (in print positions 54–59) if indicator 30 (see page 6, line 12) is off, but the word NEW to be printed instead (in print positions 57–59) if indicator 30 is on (i.e., new item this year).

6. Lines 05 and 06 on Fig. P12 provide for printing a + symbol if the cost trend is up, a - symbol if it is down, and leaving the print position blank if there has been no change in cost since the previous report. (See page 06, line 15, for setting of indicators 32 and 33.)

7. Lines 09 and 091 on Fig. P12 determines whether today's date (DATE) from the Date card or the date (TRNSDA) from the old Inventory Master card is to be printed. If there were transactions i.e., the report data is not printed while a Master card is being processed—NOI), DATE is selected; if there were no transactions (i.e., the report is based on data in the old Inventory Master card—indicator 01), TRNSDA is selected.

8. Line 10 on Fig. P12 provides for printing an asterisk in print position 120 when Quantity Available (i.e., On-Hand + On-Order) is less than the Minimum Stock Quantity (see Fig. P6, lines 13 and 14, for setting of indicator 31).

PRINTING THE GRAND TOTALS - REPORT FILE
(Fig. P12, Lines 21-17)

The line is printed at total-output time, (T in col. 15), after the last data card has been processed (LR indicator on). It must be
at total-output time, because the job is thereafter terminated if the LR indicator is on. The form is upspaced 2 lines before printing, providing 3 blank lines between the last detail line and the grand totals.

The form is advanced to channel 1 afterwards. Constants describing the fields are printed preceding the values. A fixed dollar sign is used in the edit word for Inventory Values.

INVOICING

This report utilizes the order-item cards processed in the previous program example (Pre-Billing with Inventory Control), in conjunction with sold-to and ship-to name-and-address cards. The same mail-order company is assumed, with modifications to illustrate more features.

The example is deliberately kept fairly simple, its main purpose being to provide an illustration of:

1. Printing sold-to name and address side by side, each of three lines, and each from a single card
2. Predetermined total line
3. Summary punching. The summary cards can be used for:
   a. Accounts receivable
   b. Sales report by customer
   c. Sales report by salesman
4. Card-type sequence check by sequence entry (cols. 15-16, input specifications)
5. Table lookup
ASSUMPTIONS

1. The item cards from the preceding example serve as detail cards (customer order-item cards), excluding Merchandise-return cards with 11-overpunch in col. 11. They are assumed to have been sorted by Warehouse Location and Account No. after the Pre-Billing operation.

2. The heading and detail cards have been previously match-merged so that there are no missing masters or legitimate missing details. (This match-merging could have been done on a collator, or on an MPCM.)

The card with today's date and the starting invoice number (less 1) is placed ahead of this group of cards. The deck is placed in the primary hopper of the MPCM.

3. Name and address are confined to three lines from a single card. The presence of a ship-to card is optional. When it is present, it precedes the sold-to card; when there is no ship-to card, the sold-to name and address are to be printed in both positions.

    Placing the optional ship-to card ahead improves throughput and printing of name and address can proceed during processing of the sold-to card. If the sold-to card were placed first, printing of name and address could not be commenced, when there is no ship-to card, until the first detail card is being processed; only then can the program know that no further Name-Address card (namely, ship-to card) must be awaited.

4. The blank cards, which are to become summary cards, are a separate file in the secondary hopper of the MPCM.

5. Arbitrarily, the MPCM is used for the two files; other
Card Layout

![Card Layout Diagram]

Figure 10
Card Flow

Figure 100
Invoice Layout

<table>
<thead>
<tr>
<th>IBM 407, 408, 409, 1403, 1404, 1443, and 2203</th>
<th>Point Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6 Lines Per Inch</td>
</tr>
<tr>
<td></td>
<td>1, 2, 3, 4, 5, 6</td>
</tr>
</tbody>
</table>

Figure 1000
Model 20 I/O devices can be used if the File Description Specifications are changed.

6. Stacker Selection has been arbitrarily determined thus:
   Date and Invoice-No. heading card-stacker-stacker 1;
   Name and address cards-stacker 2;
   Detail cards-stacker 2;
   Summary cards-stacker 3.

7. A discount percentage is applied to the invoice total based on a customer-type code in the sold-to card. For this, table lookup is employed.

8. Certain identifying data is repeated on overflow pages. Invoice totals are to be printed at a predetermined point on the page.

Fig. 1000 shows input and output formats. Constant headings are not printed by the program, because use of a pre-printed invoice form is usual.

In the explanations that follow for the application example, most of the obvious points will be omitted, as the reader is by this time familiar with them.

FILE DESCRIPTION SPECIFICATIONS (Fig. 11)

The input file, named INPCARDS, is associated with the primary hopper of the MFCM. It consists of one card containing the day's date and the starting invoice number (less 1) and, for each customer account number represented, contains—in this order:

One Ship-to Name-and-Address card (optional)
One Sold-to Name-and-Address card;
At least one Order-Item detail card.

A file of blank cards (named SUMCARDS), which will become the Invoice Summary cards, is to be placed in the secondary hopper of the MFCH.

The printer has been assigned the file named INVOICE.

File Description Specification

![Table]

**Figure 11**

**INPUT SPECIFICATIONS (Figure 12)**

There is only one input file, named INPCARDS, constituted of four card types.

**Date/Invoice-No. Card (Fig. 12, Lines 01-03)**

Sequence (cols. 15-16) is alphabetic, because the card appears only once, and does not fall into a sequence within each account-number group.

Stacker selection need not be specified, because 1 is the normal stacker for the primary hopper of the MFCH.

No card-type Resulting Indicator is needed; the card is never referenced, and all calculations are conditioned by indicators of the
appropriate cards.

Ship-To Card

If present, this card is to precede all others of the group; therefore, it is sequence number 01 (cols. 15-16). If present, only one is permitted; therefore, 1 is specified in col. 17. Its presence is optional; therefore, and 0 in col. 18.

Control Level 1 is assigned to customer account number—both (1) to perform end-of-invoice routines, and (2) to guard against cards out of sequence, or missing Sold-To card (see calculation specifications).

Stacker 1 is the normal stacker, and need not be designated.

Sold-To Card (Fig. 12, Lines 09-16)

Exactly one card (1 in col. 17) of this type must be present (no 0 in col. 18), and it follows the Ship-To card, but lower than for the detail cards (Fig. 13, line 01).

Different field names are used for name and address in this card: the name-and-address data from the Ship-To card (if any) is to be printed alongside that from the Sold-To card.

The same field name is used for AC NTNO in all cards because the data should be the same from all cards within the group and therefore need not be saved from card to card. If it is not the same, a control break will occur (L1 is assigned to Account No.).

Indicator 20 is utilized to recognize the first detail card of each invoice (see page 06, lines 12 and 13).

Stacker 1 need not be specified.
Order-Item Detail Cards (Fig. 13)

Our assumptions called for selecting these cards to stacker 2; therefore, a 2 is entered in col. 42 of line 01.

The field BOCARD in line 02 is specified only to provide an indicator (21) for recognition of back-order cards (11-overpunch in col. 1, making the field negative).

If the item card was to be cancelled, because of unavailability, an 11-overpunch was punched in col. 7 in the previous application example. This makes the Stock No. (line 03) negative. Indicator 22 is utilized subsequently to control operations for cancelled items.

Unit Price, among other fields, was left blank in the previous operation whenever the item could not be filled. Indicator 24 is subsequently utilized to control operations for unfilled items.

![Figure 12]
Input Specifications

CALCULATION SPECIFICATIONS (Figure 14)

Lines 01-03 cause the Sold-To name-and-address data to be moved to the corresponding Ship-To fields whenever there was no Ship-To card (i.e. the first card of the control group is a Sold-To card). At output time, this will cause the same information to be printed in the Sold-To and Ship-To areas on the invoice.

Line 031 causes the Invoice No. to be incremented during processing of the first card of each Account No. control group. (It was loaded with a value one less than the desired starting number.)

Line 04 specifies cumulation of the gross amount from each item card for an invoice total. If the item was not filled, the GRSAMT field is blank.

Line 05 causes a search through the argument table TABCOD for a code that exactly matches the Discount Code in the permanent customer Sold-To Date-and-Address card. When a match is found, indicator 23 turns on, and the discount percentage in the equivalent position of the
function table TABPRC is stored and becomes available as a calculation factor and as output-field data.

The tables are defined in File Extension Specifications—see Fig. 15.

In line 06, indicator H1 is turned on to stop the system after this card—if no Discount-Code match was achieved.

Lines 06-12 provide for the following calculations during total time following the last detail card of each invoice:

1. The invoice gross total is multiplied by the table-supplied percent of discount to establish the discount amount (line 07). (Note that half-adjustment is used, and 4 decimal positions are dropped. There are 2 decimals in INVGRS and 4 in TABPRC, since percentages less than 100 expressed as ratios fall to the right of the decimal point).

2. The discount amount is subtracted from the gross invoice amount to produce the net invoice amount (line 08).

3. The three invoice amount totals (gross, discount, net) are accumulated in three other fields, to provide grand totals (lines 09-11).

The operations in lines 07-11 are executed only when the Discount Code matched an entry in the argument table (indicator H2, and halt the system after this card, if the first card of a control group is not a Name-and-Address card (i.e., neither a Ship-To nor a Sold-To card).

Note: Since the test is made at total time (L1 in cols. 7-8), the first group will not be checked: total time is bypassed on the first card with Control Level specifications. (The test could have
been programmed for detail time instead; but our approach offers the opportunity to remind the reader of the initial total-time bypass.)

Calculation Specifications

![Table Image]

Figure 14

FILE EXTENSION SPECIFICATIONS (Figure 15)

Two tables are used in this application—one as an argument table (TABCOD) and the other as a function table (TABPRC). For convenience, the two tables are punched alternately in the same card, but this has nothing to do with the manner in which they are employed (argument or function). The table cards (in this instance, a single card) must be loaded at program-generation time.

There are only 14 codes, and all fit in one card; therefore, both the number of table entries per card and per table are the same. The code is a single character (thus, 1 in col. 42), and the percentage is 4 digits long (format XX.XX%). Since the term "percent" means "per hundred," the decimal point must be moved two positions further to the
left when multiplying by a percentage; thus, the field contains 4 decimal positions (not 2).

File Extension Specifications

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Name</th>
<th>Field Name</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>24-25</td>
<td>24-25</td>
<td>24-25</td>
</tr>
<tr>
<td>25</td>
<td>25-26</td>
<td>25-26</td>
<td>25-26</td>
</tr>
<tr>
<td>26</td>
<td>26-27</td>
<td>26-27</td>
<td>26-27</td>
</tr>
<tr>
<td>27</td>
<td>27-28</td>
<td>27-28</td>
<td>27-28</td>
</tr>
<tr>
<td>28</td>
<td>28-29</td>
<td>28-29</td>
<td>28-29</td>
</tr>
<tr>
<td>29</td>
<td>29-30</td>
<td>29-30</td>
<td>29-30</td>
</tr>
</tbody>
</table>

Figure 15

OUTPUT FORMAT SPECIFICATIONS (Figure 16)

(Refer to Card and Report Layouts.)

Note that all detail output is specified ahead of all total output.

Detail Printing on the Invoice-File (Fig. 16 and Fig. 17, Line 01-11)

This output is performed at detail time (D or H in col. 15).

INVOICE was associated with the printer (see Fig. 12, line 03).

Lines 01-04 on Fig. 16 control the printing of the first print line of the page. The Sold-To name (NAMED=Name sold), read card 2 assigned card-type Resulting indicator 02, is printed in positions 11-29; the Ship-To name (NAMEP=Name ship), read from card 01 (indicator 01) is printed in positions 58-76. Both are printed in the same line on the invoice form.

The printing at the beginning of each Account-No. group takes place as the Sold-To card is being processed (indicator 02 on); at that
time, both the ship-to and sold-to information is available, and can be printed concurrently (if 01, instead of 02 were specified, only data from the first card of the group would be available).

The names are also repeated at the top of overflow pages, at overflow-output time (indicator 0F). NLI is specified, so that the old names are not printed at overflow time at the top of one new page—followed by the new names on the next page from card type 02—when the overflow point and the end of a group coincide.

In the calculation specifications (Fig. 14, line 01), the Sold-To name was moved into the Ship-To name field if there was no Ship-To card; therefore, both names are the same in that case.

Note: If the Ship-To area on the invoice is to be left blank when there is no Ship-To card (rather than repeating the Sold-To card information), lines 01-03 on page 04 (calculation specifications) would be omitted. However, a B must then be placed in col. 39 (Blank After) of lines 04, 07, and 10 of page 06; otherwise, whenever there is no Ship-To card in a group, the data from the last preceding Ship-To card remains in storage, and will be printed.

Lines 05-07 and 08-10 provide the equivalent functions for the second and third lines of the addresses. However, the street addresses and city/state are not repeated on overflow pages.

No entry is required in cols. 17-22 of line 08, because spacing to the miscellaneous-data print line is specified in line 11. The 0 in col. 18 is entered only for compatibility with other RPG's (any entry would satisfy that requirement).

Lines 11-12 (and, as explained below, line 13) control the
conditions under which the miscellaneous data is printed above the first
detail line on the invoice.

The form is skipped to the next channel-2 punch before the
miscellaneous-data line is printed, and to the next channel-3 punch
(first detail line) thereafter.

Note: Instead of utilizing Skip/Before in specification line 11
to reach the miscellaneous-data print line (the simplest way to pro-
gram this), Skip/After—which is usually more efficient in terms of
throughput—can be used in the name-and-address specifications lines.
It requires several entries, however, because all three name-and-
address lines are printed at the start of a new customer group, but
only the name line is printed on overflow pages. The entries in cols.
17-22 (forms control) of specifications (lines 01, 02, 08, and 11)
should then read:

Line 01-$$ 01 02
Line 02-$$ 01 $$
Line 08-$$ $$ 02
Line 11-bb bb 03

The miscellaneous-data line is printed after the name and
address for a new group, and ahead of the first detail line. It is
also printed in the same position on overflow pages (when overflow
does not coincide with the end of a group); but some of the fields
are not printed (NOF) on overflow pages.

Because Customer Order No. (ORDERNO) is not available until
the first detail item card has been read, the miscellaneous-data line
must be printed after the first detail item card has been read, yet
above the regular detail data. Therefore, it is printed during processing of a detail card (indicator 03 in specifications line 12), yet before the print line for the regular detail data (see Fig. 17, lines 01-11). It is to be printed only before the first detail line (apart from overflow identification specified in line 11); therefore, the first detail card of a group must be identified. We chose to accomplish this as follows: The data for the field DSCTCO is supplied by the Sold-To card, where it is never blank. When the first detail card is processed, the DSCTCO field, therefore, contains data. (One of the possible Discount Codes in this example is 0—see Discount Table in Figure 71—but 0 is treated as non-blank in an alphameric field. DSCTCO was defined as alphameric—see Fig. 12, line 15). Indicator 20 is on only when DSCTCO is blank (see page 02, line 15); it is therefore off when the first detail card is processed.

Specifying N20 with 03 in line 12 permits the output to be performed for the first detail card, because indicator 20 is off. As the data from the DSCTCO field is transferred to the output-storage area, the Blank-After (B in col. 39) instruction causes the field to be cleared, and indicator 20 to turn on. The output controlled by the specifications in line 12 will thus never be performed again until another Sold-To card has preceded a detail card because indicator 20 remains on until data is read into the DSCTCO field again. (The entries in line 11 provide for the output at overflow time.) The field DSCTCO was chosen because its data is not needed again in the remainder of the operations for a group.

Note: An alternate approach would be:
Change all L1 specifications to L2.

Then, specify Control Level L1 for ORDRNO (Fig. 13, line 10).

In place of N20 on Fig. 16, line 12, specify L1.

The B in line 13, Fig. 16 is then not needed; nor is indicator 20 in line 15, Fig. 12 required.

This technique might be employed if the contents of all pertinent fields had to be preserved for summary punching.

Specifications lines 13-19 specify the data to be printed in the miscellaneous-data print line.

Although the field DSCTCO is not suppressed for overflow lines (no NOF entry), nothing will be printed from it, because it is blank at that time (see above).

Lines 01-11, Fig. 17 contain the specifications for printing of the item detail lines.

The ampersand symbols in the edit word for WHSLOC provide blank spaces on the invoice between the three digits.

If the order item was not filled (i.e., it was back-ordered or cancelled), the Unit Price (UNTPRI) field was left blank (in the previous operation), and indicator 24 is on (see Fig. 13, line 05). Outputs of Unit Price (UNTPRI) and Gross Amount (GRSAMT) are suppressed (N24) when these fields do not apply (i.e., they are blank, with UNTPRI used as the criterion to set indicator 24). Although the fields are blank at input, blank numeric fields are converted to zeros, and .00 would be printed if the output is not suppressed.

The QTY in line 06 pertains to Quantity Ordered; in line 07, it represents Quantity Shipped, although the data is taken from the
same field. The quantity in line 07 is therefore allowed to print only if the order item was filled (N24-UNTPRI field not blank)—it was part of the assumptions in the preceding application example that no partial fills would be made: either stock was sufficient to satisfy the quantity ordered, or the order item was not filled at all (it was then back-ordered or cancelled).

B.O. is printed in the Quantity-Shipped area on the invoice (see specifications line 08) if the order item was back-ordered and not cancelled: indicator 21 is on if the card is identified in col. 1 as a back-order card (see Fig. 13, line 02); indicator 22 is on if the order item was cancelled (see page 03, line 03). All three indicators (24, 21, N22) are needed to establish an active back order, because the item might have been previously back-ordered, and filled or cancelled in the most recent pre-billing pass (see preceding application example).

CANC is printed in the Quantity-Shipped area on the invoice (see specifications line 09) if the item was cancelled (indicator 22—see Fig. 13, line 03).

**SUMMARY PUNCHING-SIMCARDS FILE** (Fig. 17, Lines 12-20 and Fig. 18, Lines 01-05)

This output is performed at total time (T in col. 15\(^{\prime}\), at the end of an Account-No. control group (L1 in output indicators, line 12), when all totals accumulated from the cards of the group are available.

The file name SUMCARDS was associated with an output file in the secondary hopper of the MFCM (see Fig. 11, line 02). The cards are directed to stacker 3.

Lines 13-57 on page 07 contain punch, rather than interpret,
instructions, because col. 41 is blank or 0.

Lines 01-05 on page 08 contain interpreting instructions for selected fields—they are interpreting, rather than punching, specifications because col. 41 contains a print-head number (i.e., is not blank or 0).

Note 1: The interpreting feature is available only on the MFCM Model Al.

Note 2: Punching of the summary card was specified between detail and total printing to optimize throughput; generally, alternating forms printing and card punching tends to increase throughput.

TOTAL PRINTING ON THE INVOICE-INVOICE FILE
(Fig. 18, Lines 06-16)

The form is first advanced to a predetermined total line (04 in cols. 19-20, specifications line 06). Three lines of totals are then printed at total time (T in col. 15) when the L1 indicator is on (i.e., after each Account-No. group). The form is double spaced between the total lines. In specifications line 11, no entry is needed in col. 18, because forms advance before the grand-total line is specified in line 13. Zero is entered only for compatibility with other RPG's (for that purpose, any digit 0-3 is satisfactory).

Output for the second and third total lines (see specifications lines 08 and 11) is also subject to indicator 23 being on. This suppresses the discount and net amount lines when no match on Discount Code is achieved between the code in the Sold-To card and those in the argument table. While calculation of these amounts was suppressed in such case—see page 04, lines 07 and 08—.00 (not blank) would be
printed for the two amount fields (because of the format of the edit words) if output were not suppressed, and a percentage figure from an earlier LOKUP operation would be printed from TABPRC.

Whenever the total in specification line 07 is transferred to the output-storage area, the field is cleared to zero (8 in col. 39) to be ready for accumulation of the total for the next group. Note that the Blank-After instruction could not be entered on page 07 (SUMCARDS) otherwise, the field would be zero before output for printing.

In line 09, note the location of the decimal point in the edit word: in the file-extension specifications TABPRC is defined as consisting of four decimal places so that decimal alignment is correct when calculating the percentage amount. When printing the figure, however, it is to appear as a percentage again—the printing of a decimal point (like any other constant) has no connection with the location of the decimal point for arithmetic operations, as specified in the field definition.

Lines 13-16 control the printing of the grand totals at the end of the report (LR indicator on). The form is advanced to a new invoice page, and all three final totals are printed on the first line.
## Output Specifications

![Table of Output Specifications](image)

**Figure P6**

## Invoicing

![Table of Invoicing](image)

**Figure P7**
Output Specifications

<table>
<thead>
<tr>
<th>No.</th>
<th>Fieldname</th>
<th>Length</th>
<th>Output</th>
<th>Unique</th>
<th>Column</th>
<th>Constant or Edit Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INVOICE</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DATE</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure P9

CUSTOMER INVOICING

The purpose of this program is to compare the RPG invoicing program to a BAL invoicing program by means of models. The reader, who is familiar with the invoice can compare the methods, while the reader, who is not familiar with the invoice, can benefit two ways, one from the programming and the other from the invoice.

Cards

The invoice summary card contains the first billing number to be used, and the date of the billing. It has a card code of zero.

The customer address card is comparable to an accounts receivable subsidiary ledger in the file, but being processed only if a purchase was made by the customer who has purchased an item between billing dates. One card has the customer number, order number, and
order date. This is used for each customer who has purchased an item between billing dates. One card is made of each item purchased. This card has a card code of zero.

The detail item card contains a customer number, the item number, name of item, quantity of the item, the unit price of each item and the discount allowance to the customer. Later in the program, the net amount and gross amount will be punched and printed on the card.

The invoice summary card is produced at the end of the customer billing. One is produced for each customer that has ordered an item during the billing period. It will contain the customer number, the invoice number, the invoice date, and the total net amount of all items purchased. It has a card code of four.

At the end of all the billing, a new invoice card is produced, which contains only the starting invoice number. The date is punched in by the operator at the date the new invoicing is to be started. This card has a card code of zero. The last card on the deck has a cc of 5, and it is a dummy to facilitate ending the program.

Sequencing

Cards must first be sorted to be put into sequence. The customer card, the order card and the detail cards must be sorted in sequence by customer number.

Sorting

Sort on col. 1 first. This will leave three stacks of cards with a card code of 1, 2 and 3. Then remove cards with card code 2 and sort on cols. 17-16. This is the date. In case one customer ordered
more than one item during the billing period, then sort on col. 15-14. This sorts the month. The year in cols. 89-18 may also be sorted if necessary.

Now place all three stacks together. If the cards are apart to begin with only the order cards will need to be sorted by the date of the month and then the month.

All cards are then sorted on cols. 7-2, the customer number. This should place the stack ready for use. Add the invoice summary and date card to the top of the deck in front of it. Then place the two dummy cards on the back of the deck. These two cards are used to put all the cards through the machine and print and punch the final cards. The final print out will be "end of invoicing."

Every order card should have an address card behind it, followed by at least one detail card and perhaps a second order card from the same person. This person will not have an address card, but the card must be followed by at least one detail card. There may be several address cards in a row, but these are from people who have not purchased anything. They will be skipped.

Invoicing

Place all the data cards in the primary hopper.

Register 13 is being used as a base register, and the first card read is the inventory date card. It gives the starting invoice number and the date of billing. This is dumped into stacker #5. This information is moved from area 5 to area 1 (74 bytes). Then the card code only is blanked out. The inventory number, located in number 2 is packed into number 12.
The second card is read. It could be an address card or it could be an order card. If it is an order card, the information from AR5 is moved into AR3. We then compare the card code, which is Fl. It is an Fl so we branch and store to LB1.

At LB1, we will store the information such as the customer number, the order number, the month, the day and the year. The rest of the area is now cleared. This card drops into stacker number 3. BAS 9 takes us back to where we came from, and then we locate a customer address card.

At LB3, the next card is read, and this information is stored into AR2. The customer number is compared. If it does compare with the order card number, the mode of payment, located on the customer address card, is moved into number 9.

The print area is cleared (work area AR4). The name is loaded into AR4 and printed. After the next two steps are skipped, a space is taken and street address, customer, inventory number and the date are loaded into AR4. These are moved to print where they are printed and followed by a space.

We now blank out AR4-47(23) to clear the work area. By doing this we can use the same area for many things. City and State are moved to PRIN and printed, then a space is taken. The order card contents are now printed out (they are stored in LB1). Number 14, which is the customer number, is moved to AR4 and printed.

This will print out as "YOUR ORDER CARD", with customer number and dated. This is followed by a space.

At LB8, the next card is read and stored into AR3. The following
step is a no-op statement the first time through.

LB6—Compare the card code to see if it is a detail card. If it is a detail card, we go to LB4.

LB4—Process the detail item card. First check to see that there are no punches in columns 8 and 27. If no punches are located, load the quantity into number 7 and load price into 6, then multiply these two together. Now we add number 7 to number 5. They will continue to work as an accumulator to produce the total Gross Income. The Mask is moved to AR4 and the gross amount is edited. This information is moved to BRIN and printed.

The new amount is determined by packing discount into number 8 and multiplying the gross amount by the discount. The 01 instruction rounds off this amount. We may lose a penny or gain a penny by using this method, but it averages out over a period of time.

The discount is subtracted from the gross amount and the mask is moved into AR4 followed by the edit. The rest of AR4 (43 bytes), is now blanked out. The net plus gross amount are moved into AR6. This allows us to punch the information into the card.

CONT—Uses print head 2 and at AGAI (XIO AR6-7), we print on the card head the net amount and gross amount. The area is now blanked out. We check for channel 12, if yes, we go back to LB8. This means that we have now read the first invoice card, the first address card and the first detail card. Let's assume that there are no more detail cards for this person, and proceed to go back through the file again. This puts us back to where we would normally read a detail card, LB8.

LB8—A card is read and put into AR3. Now we go past the no-op
step and check to see if it is a detail card. If it is not, it must be an address card. If so, go back to LB8.

Read another card and store the information into AR3. We go past the no-op step and compare to see if it is a detail card (F3). If it is not, we pass through the next step. Is it an address card? If not we assume it is an order card. Therefore we take the BAS9 to LB2. At LB2, we store the order number, the month, the day, year and clear part of this area again. The card falls into stacker 3 and a BCR is taken back to where we came from. A compare is made to determine whether this was a detail card. If not, we proceed to LB5, and print the invoice summary information which will be the ending statement for this customer.

A mask is moved into AR4, and the gross amount is edited and printed. After printing, a space is taken, and the percent of discount is entered.

The discount (no. 10) is packed into number 8. The percent is multiplied times the amount to produce the net amount. A round is taken and the mask is moved into PRIN and edited, after which, we print and space and then blank the rest of AR4.

We take the gross amount and subtract the net amount from it, and move the mask to AR4. Then we edit the net amount into AR4. The net area is cleared by subtracting it from itself. This information is printed and spaced.

Read a card and store this information into AE2. Compare the customer number (let's say they do compare). If they did not compare, we would go back to LB3 and cycle until we did find a number that does
compare, or we would go back and cycle until we came to the customer number that has a higher number than the one located in LB2. If this happens, we branch to SEQR, which is an out of sequence routine, which halts the program.

LB3—Read a card from the order file. At LB3, the information from this card is put into AR3 and we go past the no-op. We now compare the card to see if it is a detail card. If it is, we branch to LB4.

LB4—A compare is made of columns 8–27 to be sure there are no punches. However, this time there is a punch in the card. Branch off to PCHF, which is a read format error routine. This card is checked again to make sure it is a detail card. (We assume it is.) Switch with the MVI instruction EOFQ–1 and move 00 into this condition code. The next instruction moves a blank into AR4 followed by a clear with a MVC instruction.

The print error test (number 13) is moved into AR4+50 for 24 bytes. This error message will print out on the paper as a "punch error" (in the item card). Then blank out the text-error message and place this card into stacker number 2.

Read the next card and place the information into AR3, check the customer number to see if it is a new customer. If it is not a new customer, we go back to CB9 and read another card. We continue this until all of a customer's cards are used up. This is indicated by the appearance of a different customer number.

If a new number appears, we move down to the next MVI instruction and change the card code that changed before. This time it is changed to 15. Then we unconditionally branch back to LB3–4; the
instruction is a BAS9 to LB1. Store the customer number, order number, day and year, then clear the area. This card will go to stacker number 3. The BCR is taken back to LB3.

At LB3, locate a customer's address card to match this order number. If the numbers do match, print the invoice header information, order card contents.

Read a card from the order file, and go to LB4 for the processing of the detail item card. Check for punches, if no, go on down and print the information, and punch and write on the card. The TIOB instruction checks for the overflow which is "43". Check for channel 12. If we were there, go to OVF1, which is the channel 12 condition routine. The MVI 15 into LB8-11 moves the unconditional branch condition into LB8+11. We then branch unconditionally back to LB11 - 4. This instruction is a RC 15 to LB8.

At LB8, we read a card from the order file. The information is moved into AR3. Then BC15 to LB11. The zero condition code has just been changed to a 15 by the MVI instruction in the overflow area (OVFL).

At LB11, we immediately make LB8+11 a zero condition code again. Compare customer numbers, if not equal, branch to LB12 and at LB12 (BASF) to LB2.

LB2—Store the order number, month, day and year, and then clear the area. Put this card into stacker 3, and branch back to the statement following the BAS 9 where we just came from (LB12).

The next instruction is a CLL statement, whereby, we compare to see if this is a detain card. If it is not a detain card, we branch to LB5.
LB5 is an area where the ending routine is effected for this customer. We print the invoice summary information, produce the summary card, increase the invoice number, and now branch back to LB3.

The dummy card is read at LB3, which is a 5 in column 1, and in columns 7-27 is punched "end of invoicing". This information is stored into AR2. We then compare the customer numbers. The customer numbers do not compare because they are all blanks. We drop through the next two routines. Blank out the mode of payment because this area is blank.

The net amount is to be saved so it can be punched into the invoice summary card. The area is cleared and mode is moved. Amount of days, percent of discount and the days net are moved into mode.

The last line is printed out as Mode of Payment 20 days 2% discount OK 30 days Net. Naturally the figures will change with each customer. Check for channel 12, if not, go down and print and skip to column 1 and blank out more of AR4 to clear the area.

Move the card number, which is NO. into AR2 number (24). The customer number, inventory number, the date and the net amount (2), and put AR2 into AR7.

We take the BAS 8 and branch and store to punch.

This action produces a new invoice summary card, which has a card code of 4. A return is now made from the punch routine.

Move AR12, which is the invoice number, into CPL1. Move area 13, which is an inventory date into CPL 2, clear part of AR2, and move customer number into CPL4 and BAS 8 to HEAD, where we write on the card.

The card will contain customer number, invoice number, invoice date and net amount. This card will then be placed into stacker 5.
To increase the invoice number, we take number 19, which is the number 1, and add it to 12, which is the invoice number. Move in the mask and edit the number, and go back to LB3.

At LB3, we are going to try to locate the customer address card number to match the order number, that we have stored in LB2.

The header information is printed by first printing the name, but this time it is not the same, it is the "END OF INVOICING". This is printed out after which we come to the CALL we have been passing up all the time during the program. We check to see if it is a card code of 5. We assume it is so we go to EOFR, which is the end of the file routine.

EOFR—Move the new invoice number, which is at number 2 to AR4+35. Reset the switch at PNCH-5. This cuts down the amount of information that is going to be printed on the card from 16 down to 8. Now we take the next MVI instruction which moves a 00 into PUER3. This changes the hopper code from 4 to 5. This card will now drop into stacker number 5.

We then move a card indicator code into AR4+34. Move AR9 into AR7, which is the invoice number. We then branch to PNCH and pick out this last card. Only the invoice summary number will be punched. The date will have to be punched in by the operator at the next billing period.

After number 1 is selected to print this same information, the final action is the halt.
Invoicing—Basic Assembler

Figure INVO-1
Figure INVO-3
Invoicing—Basic Assembler

```
0444  0111111100
044C  0111111110
0452  0111111110 4111111110
045A  0111111110 4111111110
045E  0111111110 4111111110
0462  0111111110 4111111110
046A  0111111110 4111111110
0472  0111111110 4111111110
047A  0111111110 4111111110
047E  0111111110 4111111110
0482  0111111110 4111111110
048A  0111111110 4111111110
048E  0111111110 4111111110
0492  0111111110 4111111110
049A  0111111110 4111111110
049E  0111111110 4111111110
04A2  0111111110 4111111110
04A6  0111111110 4111111110
04AC  0111111110 4111111110
04B2  0111111110 4111111110
04B4  0111111110 4111111110
04B6  0111111110 4111111110
04B8  0111111110 4111111110
04BE  0111111110 4111111110
04C5  0111111110 4111111110
04CA  0111111110 4111111110
04CE  0111111110 4111111110
04D2  0111111110 4111111110
04D4  0111111110 4111111110
04D6  0111111110 4111111110
04DF  0111111110 4111111110
04F4  0111111110 4111111110
04F6  0111111110 4111111110
04F8  0111111110 4111111110
04FC  0111111110 4111111110
04FEE  0111111110 4111111110
0504  0111111110 4111111110
0506  0111111110 4111111110
0508  0111111110 4111111110
050A  0111111110 4111111110
050C  0111111110 4111111110
050E  0111111110 4111111110
0514  0111111110 4111111110
0516  0111111110 4111111110
0518  0111111110 4111111110
051A  0111111110 4111111110
051C  0111111110 4111111110

Figure INVO-04
```
Figure INVO-05
Invoicing—Basic Assembler

Figure INVO-06
This section will aid in the preparation of the work required for the Swanson study, since a disk operation or sorts is assumed. The main topics are presented on the next page, and are presented as File Description Specifications. The two main divisions of the section are:

a. Sequential Disk Files

b. Indexed Sequential Files

SEQUENTIAL DISK FILES

Pre-sorted records are read from another external file (or disk) and written in sequence within the extents on the disk as fixed length records, either blocked or unblocked. The Job Control program is presented the actual location of the file on the disk at the time the file
is loaded. The file label is written in the VOTC (volume table of contents). The label locates, identifies and protects the files.

File Description Specifications

Figure S-0  RPG Coding for Disk Operations
This RPG program extracts fields from types of cards and writes a sequential field of ninety character records per block. A numeric field is packed on a disk that was unpacked on cards. A program constant "M" is placed in the first byte of a record. The "M" is a constant which is not contained in data cards.
Output-Format Specifications

Figure S-1 (continued). RPG Coding
Sequential File - Loading the File, Part 2 of 1

SEQUENTIAL FILE - SEQUENTIAL RETRIEVAL

File Description Specifications

Figure S-2 Sequential File - Sequential Retrieval

Sequential Retrieval is accomplished by letting column 28, column 31 and column 32 remain blank, but the number of extents as indicated by column 68-69 must agree with the number assigned when the file was written.
SEQUENTIAL FILE - SEQUENTIAL RETRIEVAL
WITH MATCHING RECORD LOGIC

File Description Specifications

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input Specifications

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-3. RPG Coding Sequential File
Sequential Retrieval with Matching Record Logic

This permits selective processing of from two through nine input files. The files must be in the same collating sequence, but may be on any input device. The matching fields are assigned on the Input Specifications in columns 61 and 62 as matching fields M1 through M9. The files are processed first by selecting them in the order they are entered on the Input Specifications. The internal matching record indicator controls the processing of matching fields. The MR is tested on the calculation specification and processing is modified by its setting.
SEQUENTIAL FILE - RANDOM RETRIEVAL
WITH ADDROUT OPTION

File Description Specifications

Figure S-4. Sequential File
Random Retrieval with ADDROUT Option
File Description Specifications

It is not necessary to require the whole file to be updated if the record lengths are not modified, however if the length is to be modified, the entire file must be copied to an output-file.
SEQUENTIAL FILE - UPDATING AND ADDING RECORDS TO THE FILE (2 pages)

File Description Specifications

<table>
<thead>
<tr>
<th>File Description Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential File Updating and Adding Records to the File, Part 1 of 2</td>
</tr>
</tbody>
</table>

Input Specifications

<table>
<thead>
<tr>
<th>Figure S-6, Sequential File</th>
</tr>
</thead>
</table>

Updating and Adding Records to the File, Part 1 of 2
Updating and Adding Records to the File (con't.)

Calculation Specifications

Output-Format Specifications

Figure S-7. RPG Coding. Sequential File

Updating and Adding Records to the File. Part 2 of 2

The File must be rewritten, but records can be updated during the same run. The files are specified as input and output, not as update, since update files must not include additions and deletions.
II  INDEXED SEQUENTIAL FILE

File Description Specifications

<table>
<thead>
<tr>
<th>Card Type</th>
<th>File Format</th>
<th>Disk Volume</th>
<th>Disk Cylinder</th>
<th>Record Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDG</td>
<td>Indexed</td>
<td>Disk 0</td>
<td>Cylinder 0</td>
<td>KEY</td>
</tr>
<tr>
<td>FDG</td>
<td>Indexed</td>
<td>Disk 1</td>
<td>Cylinder 1</td>
<td>KEY</td>
</tr>
</tbody>
</table>

CYLINDER OVERFLOW AREA
XTENT CONTROL STATEMENTS
- 3 PRIME DATA AREAS
- CYLINDER INDEX
- INDEPENDENT OVERFLOW AREA

INDEXED - SEQUENTIAL FILE
RECORD KEY

Figure S-8. RPG Coding

Indexed-Sequential File - Loading the File

Records are fixed length blocked or unblocked. As each file is written, indexes are created to provide a reference to records on each track or cylinder. The file is an output file designated by a 0 in column 15. There is a K in column 31 and an l in column 32. Columns 29-30 and 35-38 specify the length and location of the key field. There are two additional extents for the prime data area and one extent for the independent overflow area entered in columns 68-69. The user provides information about these with an extent control statement to the Job Control program when the load operation is enacted.
INDEXED SEQUENTIAL FILE - SEQUENTIAL RETRIEVAL

File Description Specifications

Retrieval is similar to retrieval of sequential files, and additions written in overflow areas are retrieved in sequential order. When multiple input files are specified, the order of processing is determined by the sequence they are entered on the Input Specifications. Coding follows the basic pattern for coding a disk file, plus a K in column 31 and an I in column 32. This indicates indexed-sequential organization. Column 28 is blank, and causes the entire file to be processed.

INDEXED SEQUENTIAL FILE - SEQUENTIAL RETRIEVAL
WITH MATCHING RECORD LOGIC

File Description Specifications
Input Specifications

INDEXED SEQUENTIAL FILE - SEQUENTIAL RETRIEVAL
WITH AN RA FILE OF LIMITS

The RA (Record Address) is used to retrieve a segment of an indexed sequential file. The RA File defines the low and high limits of data to be processed. Two keys are contained in each record of the RA File (the lower limit and the upper limit). The record with a key equal to, or higher than, the lower limit is retrieved; all others above it follow in sequence up to, or equal to, the upper limit.

Only "keys" are contained in the RA file, and their purpose is to retrieve records. No reference is made to it on the Input Specifications so fields for update purposes can't be defined.

All information for the RA comes from the File Description and File Extension Specifications. An addition of a 1 in the "MODE OF PROCESSING" (column 28) of the File Extension Specifications tells that the RA File is related to the Indexed-Sequential File.
## File Description Specifications

![Diagram of File Description Specifications]

## File Extension Specifications

![Diagram of File Extension Specifications]

**Figure S-11. Indexed-Sequential File Sequential Retrieval with RA File of Limits. Part 1**

## Input Specifications

![Diagram of Input Specifications]

**Figure S-11. Indexed-Sequential File Sequential with RA File of Limits. Part 2 of 2**
INDEXED-SEQUENTIAL FILE - RANDOM RETRIEVAL WITH CHAINING FILE

The "Chaining" technique is also used with the indexed sequential file. This method of random retrieval uses keys or chaining fields. They also serve as the link between two files. Each record in a chaining file contains the key of a record to be randomly retrieved from the chained file, which must be organized as an indexed-sequential file.

See Figure S-12. Chaining is encountered, in addition to the entries for the indexed sequential file:

1. AC is included in the File Description (column 16) of the File Description.
2. An R is placed in Mode of Processing (column 28).
3. An I is placed in Extension Code (column 39), to reference the File Extension Specifications. It relates the file as the chaining file (columns 11 to 18) to the disk file, which is the chained file (columns 19-26) by the chaining field of C1 (columns 9-10).

Indexed-Sequential File
Random Retrieval with Chaining File (con't.)

File Description Specifications
File Extension Specifications

Input Specifications

Key or Chaining Field

Figure S-12. Indexed-Sequential File
Random Retrieval with Chaining File

INDEXED-SEQUENTIAL FILE - RANDOM RETRIEVAL WITH
ONE CHAINING FILE FROM THREE CHAINED FILES

File Description Specifications
As depicted by Figure S-13, another method of using the chaining technique is to retrieve records from a chained file. Then use a field in those records to link to another file. The chained file then becomes a chaining file to the second chained file.

The chained file is unique type of RPG file. Records in this file are not retrieved until total time in the RPG processing cycle, while other files are read before total time. As a result, a chained file cannot contain control fields or matching fields. It is the only file which can be updated at total time.
INDEXED-SEQUENTIAL FILE - RANDOM RETRIEVAL
WITH CHAINED FILE AS A CHAINING FILE

File Description Specifications

File Extension Specifications

Input Specifications

Figure S-14. RPG Coding. Indexed-Sequential File
Random Retrieval with Chained File as a Chaining File
ACTIVITIES

The final activities section presents a culmination of the book. The material was presented by models and methods for accomplishing these models.

If the reader will review Section I and the Preface, he will notice that this book could be used for a complete year.

This (year) can be easily accomplished by going back to the Swanson Study, and writing programs for the functional areas of the corporation.

It is suggested that the reader program for Financial Control first. This includes:

1. General Ledger and Budget Accounting

   This activity reports all income, expenses and budget information to Management Planning.

2. Accounts Receivable

   This activity accounts for all customer receivable information resulting from sales. Here customer records are maintained, reflecting credit status, statement of customer accounts, and customer receipts. The income is reported to General Ledger Accounting.

3. Cost Accounting

   This activity accounts for and reports costs of production which includes material costs and labor costs.

4. Labor and Payroll

   This activity accounts for payroll for all employees. A report is also made to General Ledger for payroll and
labor expenses. Cost Accounting also receives a report based on labor tickets from Production Planning and Control.

5. Accounts Payable

This activity accounts for, and reports payments for, goods and services received. Expenses are reported to General Ledger Accounting.
ADD REGISTER

$$\begin{array}{c|c|c}
AR & R_L & R_2 \\
\hline
[RR] & 1A & R_1 \\
0 & 78 & 1112.15 \\
\end{array}$$

The second operand is added to the first operand, and the sum is placed in the first operand.

Both operands and the sum are 16-bit integers.

Condition Code:

0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero

AR10,11
ADD HALFWORD

```
<table>
<thead>
<tr>
<th>4A</th>
<th>R_L</th>
<th>X_2</th>
<th>D_0</th>
<th>D_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>78</td>
<td>1112</td>
<td>1516</td>
<td>1920</td>
</tr>
</tbody>
</table>
```

The halfword second operand is added to the first operand, and the sum is placed in the first operand (register).

1. The second operand is a 16-bit signed integer.
2. The operand must be on a halfword integral boundary.
3. The first operand is a 16-bit signed integer.
4. The sum is a 16-bit signed integer.

Condition Code:

0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero

AH 8,HALF
A branch to the address specified in the second operand is taken whenever the condition code matches a condition specified in the first operand (M).

To code this instruction:

1. Place the mask value corresponding to the desired condition.
   Place the mask value corresponding to the desired condition code in the first operand.

   **Condition Code 0 1 2 3**
   **Mask Value 8 4 2 1**

   **Example:**
   A. Desired Condition Code is 1
   B. Coding is BC4, TOPL

   2. To test for more than one condition code, place the sum of the mask values corresponding to the desired condition codes in the first operand.

   **Example:**
   A. Desired condition codes are 0 and 2.
   B. Coding is BC10, BRANCH
Note: Either condition code 0 or condition code 2 will cause a branch.

3. When the first operand is 15, the branch is always taken (unconditional branch).

4. When the first operand is 0, no branch is taken (a no-operation [no-op] equivalent).

COMPARE HALFWORD

\[
\begin{array}{cccccc}
\text{CH} & R_1 & D_2 & \{X_2'B_2\} & \text{[RX]} \\
49 & R_1 & X_2 & B_2 & D_2 \\
0 & 78 & 1112 & 1516 & 1920 & 31
\end{array}
\]

The first operand is compared algebraically with the halfword second operand, and the result determines the setting of the condition code.

1. Both operands are 16-bit signed integers.

2. The second operand must be on a halfword integral boundary.

Condition Code:

0 operands are equal
1 First operand is low
2 First operand is high
3 --

CH 8, LIMT
COMPARE DECIMAL

\[
\text{CE } D_1(L_1B_1) \cdot D_2(L_2B_2)
\]

The first operand is compared algebraically with the second operand, and the result determines the setting of the condition code.

1. Both operands are in packed decimal format.
2. Comparison proceeds from right to left taking into account the sign as well as all the digits of each field.
3. The operand 2 field must not be longer than the operand 1 high-order zeros.
4. Plus zero and minus zero compare equal.

Condition Code:

0 Operands are equal
1 First operand is low
2 First operand is high
3

CP YTDF,AMT
BC 0,NOTX
EDIT

The format of the second (source) is changed from packed to zoned and is edited into the first operand.

1. The second operand (source) must be in packed format.
2. Editing proceeds left to right, one character at a time.
3. The edited result replaces the pattern.

Condition Code:

0 Result is zero.
1 Result is less than zero.
2 Result is greater than zero.
3 —

ED OLPR-1(19), PRIN

LOAD HALFWORD

The halfword second operand must be on a halfword integral boundary.

Condition Code:

The code remains unchanged.

LH 11,HALF
The second operand is placed in the first operand location.

**MVC (Move Characters)**

1. The bytes are moved one at a time in each field.
2. Movement is left to right.
3. The number of bytes is determined by the implicit or explicit length of the first operand.

**MVC (Move Immediate)**

1. One byte of immediate data is stored in the first operand location. Immediate data supplied by the instruction itself, and in this case, is the second operand.

```
MVI SW,X"01"
MVI OLD,NEW
```
The numeric portion (low-order four bits) of each byte in the second operand are placed in the numeric portion of the corresponding bytes in the first operand.

1. The number of bytes in the operation is determined by the implicit or explicit length of the first operand.
2. Movement is from left to right through each field.
3. Movement is one byte at a time.
4. Zones remain unchanged.

Condition Code:

The code remains unchanged.
The signed or unsigned number in the zoned format, in the second location is changed to packed format and stored in the first operand location.

1. The fields are processed from right to left.
2. If the first operand field is too long, it will be filled with high-order zeros.
3. If the first operand field is too short, any remaining high-order digits in the second operand will be ignored.
4. The maximum size of the second operand (zoned field) is 16 bytes.

Condition Code:
The code remains unchanged.

PACK SGND, ZONE
STORE HALFWORD

The general register specified in the first operand is stored at the second operand halfword location.

The second operand must be on a halfword integral boundary.

Condition Code:

The code remains unchanged.

STH R1, D2 (X2,B2) [SS]

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>X2</th>
<th>B2</th>
<th>D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td></td>
<td>78</td>
<td>11</td>
<td>20</td>
</tr>
</tbody>
</table>

STH 8, BUTE
SUBTRACT HALFWORD

The halfword second operand is subtracted from the register specified in the first operand, and the difference is placed in the register.

1. The second operand is a 16-bit signed integer.
2. The second operand must be on a halfword integral bound.
3. The first operand is a 16-bit signed integer.
4. The difference is a 16-bit signed integer.

Condition Code:

0 Difference is zero.
1 Difference is less than zero.
2 Difference is greater than zero.

SH 10, HALF
The second operand is subtracted from the first operand, and the difference is placed in the first operand.

1. Both operands must be in packed format.
2. The difference is in packed format.
3. If the first operand is too short to contain the difference, overflow occurs, and the carry is lost.
4. If the second operand is shorter than the first, subtraction will take place normally.
5. A field may be subtracted from itself.
6. The second operand must not be longer than the first or a program error stop occurs.

Condition Code:

0 Difference is zero
1 Difference is less than zero.
2 Difference is greater than zero.
3 Overflow.

SP PAWT, PINT(3)
MOVE WITH OFFSET

![Diagram of MOVE with Offset](image)

The second operand is placed to the left of, and adjacent to, the low-order four bits of the first operand.

1. The fields are processed right to left.
2. If the second operand is shorter than the first operand, it is extended with high-order zeros.
3. If the first operand field is shorter than the second operand, the remaining information is ignored.

Condition Code:
The code remains unchanged.

MVO BDSN, GOSN(6)
MOVE ZONES

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
D_3 & L & B_1 & D_1 & B_2 & D_2 \\
\hline
0 & 78 & 1516 & 1920 & 3132 & 3536 \\
\hline
\end{array}
\]

The high-order 4 bits (zone portion) of each byte of the second operand are placed in the first operand field.

1. Movement is from left to right.
2. Movement is one byte at a time.
3. The number of zone portions moved is determined by the implicit or explicit length of the first operand.

Condition Code:

The code remains unchanged.

\[
\text{MVZ ZNUM+4(L),ZNUM+3}
\]
MULIPLY DECIMAL

The first operand (multiplicand) is multiplied by the second operand (multiplier), and the product is placed in the first operand location.

1. Both the multiplicand and the multiplier must be packed.

2. The product is in packed format.

3. The length of the first operand in bytes must be equal to or greater than the number of bytes required to contain all of the multiplicand plus the total number of bytes in the multiplier (second operand) field.

Example:

Multiplier - XXXX 0X XX Xs (3 bytes)
Largest Multiplicand - XXXXXXXX XX XX XX XS (4 bytes)

The product field length must then be 7 bytes (3-4) or larger in length XX XX XX XX XX XS

4. The multiplier may not exceed 15 digits and sign (8 bytes).

5. The maximum product size is 41 digits and sign (16 bytes).

Condition Code:

The code remains unchanged.

MP PINT,PRAT
OR IMMEDIATE

\[
\begin{array}{cccc}
0 & 1 & 0 & 1 \\
96 & 12 & B_1 & D_1 \\
0 & . & 78 & 15 & 16 & 19 & 20 & 31
\end{array}
\]

The first and second operands are examined on a corresponding bit by bit basis.

1. If either or both of the corresponding bits are ones, the result is a one and replaces the bit in the first operand.

2. If both bits are zeros, the result is zero and replaces the operand.

3. The second operand is one byte (8 bits) of immediate data which operates with one byte of data at the first operand storage location.

Condition Code:

0 Result is zero.

1 Result is not zero.

01 OPRD,X"08"
BRANCH AND STORE

BASR $R_1$, $R_2$  \[\text{[RR]}\]

<table>
<thead>
<tr>
<th>0D</th>
<th>$R_1$</th>
<th>$R_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>78</td>
<td>1112</td>
</tr>
</tbody>
</table>

BAS $R_1$, $D_2$, $X_2$, $B_2$, $D_2$  \[\text{[RX]}\]

<table>
<thead>
<tr>
<th>4D</th>
<th>$R_1$</th>
<th>$X_2$</th>
<th>$B_2$</th>
<th>$D_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>78</td>
<td>11</td>
<td>12</td>
<td>15</td>
</tr>
</tbody>
</table>

The address of the next sequential instruction (nsi) is stored in the first operand, and a branch is taken to the address stored in the second operand.

BASR (Branch and Store Registers)

1. If the second operand is 0 (zero), no branch is taken.

Condition Code:

The code remains unchanged.

BEGIN BASR 11.0
The packed format second operand location is changed to signed zoned format and is placed in the first operand location.

1. The fields are processed from right to left.

2. If the first operand field is too long, it will be filled with high-order zeros.

3. If the first operand field is too short, any remaining high-order digits will be ignored.

4. The maximum size of the first operand (zoned field) is 16 bytes.

Condition Code:

The code remains unchanged.
COMPARE LOGICAL

\[
\begin{array}{cccc|c}
\text{CLI} & D_1 \cdot B_1 & 12 & [s1] \\
65 & 78 & 15 & 16 & 19 & 20 & 31 \\
\end{array}
\]

\[
\begin{array}{cccc|c}
\text{CLC} & D_1 & (L \cdot B_1) & 0 \cdot D_2 (B_2) & [ss] \\
D_5 & L & B_1 & D_1 & B_2 & D_2 \\
0 & 78 & 15 & 16 & 19 & 20 & 31 & 32 & 35 & 36 & 47 \\
\end{array}
\]

The first operand is compared with the second operand, and the result is indicated in the condition code.

1. Comparison is binary (that is, bit by bit).
2. Comparison proceeds from left to right.
3. Comparison ends as soon as an inequality is found.

CLI only:

One byte in the storage location specified by the first operand is compared with one byte of immediate data.

CLC only:

The number of bytes to be compared is specified by the implicit or explicit length of the first operand.

Condition Code:

0 Operands are equal.
1 First operand is low.
2 First operand is high.
3 

CLI CODE, "E"

CLC NAME, "SMITH"
ADD DECIMAL

\[ AP \ D_1 (L^1B_1) \ldots D_2 (L_2 \ B_2) \]

| FA | L_1 | L_2 | B_2 | \{ D_1 | B_2 \} | D_2 |
|----|-----|-----|-----|--------------|-----|
| 0  | 78  | 11  | 1516| 1920 3132 3536 47 |

The second operand is added to the first operand, and the sum is placed in the first operand storage location.

1. Both operands must be in packed format.
2. The sum is in packed format.
3. If the first operand is too short to contain the sum, overflow occurs and the carry is lost.
4. If the second operand is shorter than the first operand, addition will take place normally.
5. A field may be added to itself.
6. The second operand may not be longer than the first or an error stop occurs.

Condition Code:

0 Sum is zero.
1 Sum is less than zero.
2 Sum is greater than zero.
3 Overflow
TEST UNDER MASK

The state of the first operand bits selected by a mask (second operand) is used to set the condition code.

1. 1-8 bits may be tested.
2. The mask is one byte (8 bits) of immediate data (second operand).
3. A mask bit of one indicates that the corresponding storage bit is to be tested.
4. A mask bit of one indicates that the corresponding storage bit is to be ignored.

Condition Code:

0 Selected bits or mask, all-zero.
1 Selected bits mixed zero and one.
2 --
3 Selected bits all-one.

\texttt{TM BUTE,X"FF"}

\texttt{BC 4, NWRT}
ZERO AND ADD

The storage location specified by the first operand is cleared to zero, and then the second operand data (packed format) is added to the first operand.

1. The length of the second operand must not be greater than the first operand.

2. A negative zero balance will be made positive.

Condition Code:

0 Result is zero.

1 Result is less than zero.

2 Result is greater than zero.
Absolute Address: A pattern of characters that identifies a unique storage location or device without further modification. Synonymous with machine address.

Access Method: Any of the data management techniques available to the used for transferring data between main storage and an input/output device.

Accumulator: A register in which the result of an arithmetic or logic operator is formed.

Address: An identification, as represented by a name, or number for a register location in storage, or other data source or destination. Loosely, any part of an instruction which specifies the location of an operand for the instruction.

Address Calculation: A calculation performed on a key so that the result is an address.

Address Modification: The process of changing the address part of a machine instruction by means of coded instructions.

Address Register: A register that stores an address.

Alias: Another name by which a file is known.

Alphameric: A generic term for alphabetic letters, numerical digits and special characters.

Ascending Order: The word with the lowest key appears in the cell of the lowest number.

Assembler: A program that assembles. See assemble——A program which prepares an object language program by producing absolute or relocatable machine code from a source program of statements containing symbolic operation codes and symbolic operands.

Assemble: To prepare an object-language program from a symbolic language program by substituting machine operation codes for symbolic codes and absolute or relocatable addresses for symbolic addresses.

Auxiliary Function: A function that is either control or data oriented but does not cause data transmission.

Base Register: A register used for addressing purposes.

Basic Assembler Language: A symbolic language for the writing of source programs.

Basic Assembler Program: A program used to translate source programs written in basic assembler language into machine language.
Basic Monitor: The main control program of the DPS. Available in a card of disk version. Resident in core storage when control is required. Loads programs into core storage and causes their execution.

Bifurcating: No vortex is higher than three, for an absence only root-like, simple linking nodes, and one root are permitted.

Bifurcating Cell: Contains two pointers.

Binary: 1. Pertaining to a characteristic or property involving a selection, choice, or condition in which there are two possibilities.
2. Pertaining to the number representation system with a base of two.

Binary Code: A code that asks use of exactly two distinct characters, usually 0 and 1.

Binary-Coded Character: One element of a notation system for representing alphanumeric characters such as decimal digits, alphabetic letters, punctuation marks, etc., by a fixed number of consecutive binary digits.

Binary-Coded Decimal: Pertaining to a decimal notation in which the individual decimal digits are each represented by a binary code group: e.g., in the 8-4-2-1 coded decimal notation, the number twenty-three is represented by 0011, in binary notation, twenty-three is represented as 10111.

Binary Digit: A character used to represent one of the integers smaller than the Radix 2.

Bit: A binary digit.

Bit: The smallest unit of information in System/360. It can have either of the two binary values: zero or one.

Bit Extraction: Mapping by extracting specified bits from a key and then compressing.

Binary-to-Decimal Conversion: Conversion of a binary number to the equivalent decimal number; i.e., a base-two number to a base-ten number.

Blank: One of the characters in a character.

Blank Character: Any character or characters used to produce a character space on an output medium.

Blocks: 1. A physical set of records grouped for the purpose of conserving tape; or disk storage space; or increasing the efficiency of access or processing.
2. A group of bytes, transferred to or from a physical unit device in one operation, may contain one or more logical records.

Branch: To depart from the normal sequence of executing instructions in a computer. A machine instruction that can cause a departure as in (1). Synonymous with "transfer".

Buffer: Area in memory which holds data brought in from an input device.

Byte: A sequence of adjacent binary digits operated upon as a unit.

Byte: The basic unit of information in System/360. Every byte consists of eight bits each having a value of zero or one, (see bit).

Card Code: The combination of punched holes which represent characters (letters, digits, etc.) in a punched card.

Card Column: One of the vertical lines of punched positions on a punched card.

Card Field: A fixed number of consecutive card columns assigned to data of a specific nature.

Card Image: One-to-one representation of the contents on a punched card.

Card Punch: A device to record information in cards by punching holes in cards to represent letters, digits, and special characters.

Card Reader: A device which reads and translates into internal form the holes in punched cards.

Card-Resident-System: Consists of the card control programs, Basic Monitor, Job Control, and initial Program Loader. Used for the execution of object programs contained in punched cards.

Card Stacker: A mechanism which stacks cards in a pocket after they pass through a machine.

Cell: Continuous locations where a record is stored.

Character: One of a set of elementary symbols which may include decimal digits 0 through 9, the letters A through Z, punctuation marks, and any other symbols acceptable to a computer for reading, writing or storing.

Character Set: A list of characters acceptable for coding to a specific computer or input/output device.

Clear: To put a storage or memory device into a prescribed state, usually that denoting zero or blank.
Coded Decimal: A type of notation in which each decimal digit is identified by a group of binary ones and zeros.

Column Binary: Pertaining to the binary representation of data on punched cards in which adjacent positions in a column correspond to adjacent bits of data, e.g., each column in a 12 row card may be used to represent 12 consecutive bits of a 36 bit word.

Command: An instruction in machine language.

Communication: The process of transferring information from one point, person, or equipment to another.

Communication Region: An area of the Basic Monitor. Contains date, storage capacity specification, UPS1 byte, user areas 1 and 2, and program-name area. Provides for inter-program and intra-program communication.

Compiler: A compiler is a program which prepares an object language program.

Component: A basic part, an element.

Compound Key: Combining several fields of the record for either ordering or searching.

Computer: 1. A device capable of solving problems by accepting data, performing prescribed operations on the data, and supplying the results of these operations on the data. Various types of computers are calculators, digital computers and analog computers.

2. In information processing, usually an automatic stored-program computer.

Computer, Hexadecimal Number System: A number system using the equivalent of the decimal number sixteen as a base.

Computer Instructions: Same as machine instruction.

Constant: A fixed or invariable value or data item.

Control Field: A group of contiguous bytes that are within a data record. The sort, or merge of the records, is based on the collating sequence as applied to these bytes.

Control Statement: A punched card that contains information to tailor a program to the requirements of the user.

Convert: To change the representation of data from one form to another; e.g., to change numerical data from binary to decimal or from cards to tape.
Core-Image Directory: A table on the system disk pack containing the addresses and sizes of the program and/or program phases in the core-image library.

Core Library: A disk area containing the job control program, other IBM-supplied programs (except the Basic Monitor), and user's problem programs. Permits retrieval of programs and/or phases by the Basic Monitor.

Core-Image Maintenance Program: A service program. Updates the core-image library and directory. Is used to add and/or delete phases.

Core Storage: A form of high speed storage using magnetic cores.

Counter: A device such as a register or storage location used to represent the number of occurrences of an event.

Cycle: 1. An interval of space or time in which one set of events or phenomena is completed.

2. Any set of operations that is repeated regularly in the same sequence. The operations may be subject to variations of each repetition.

Cylinder: All the marks which can be accessed without moving the READ/WRITE Heads.

Data: Any representation, such as character quantities, to which meaning might be assigned.

Data Conversion: The process of changing data from one form of representation to another.

Data File: A collection of related records organized in a specific manner, for example, a payroll file (one record for each employee, showing his rate of pay, deductions, etc.) or an inventory file (one record for each inventory item, showing the cost, selling price, number in stock, etc.).

Data Management System: A data management system deals with records in a file. The records contain information called elementary data items, which are the object of processing.

Data Processing: A systematic sequence of operations performed on data.

Data Processing System: A network of matching components capable of accepting information, processing it according to a plan, and producing the desired results.

Data Structure: The arrangement and interrelation of records in a file form; a data structure.
Decimal: 1. Pertaining to a characteristic or property involving selection, choice or condition in which there are ten possibilities.
   2. Pertaining to the number representation system with a radix of ten.

Deck: A collection of punched cards.

Decimal-to-Binary Conversion: The conversion of a decimal number to the equivalent binary number, i.e., a base-ten number to a base-two number.

Decision: A determination of future action.

Decision Instruction: An instruction that effects the selection of a branch of a program, e.g., a conditional branch instruction.

Decrement: The quantity by which a variable is decreased.

Decision Box: A flow-chart symbol whose interior contains the criterion for decision or branching.

Diagram: A schematic representation of a sequence of operations or routines.

Diagnostic: Pertaining to the detection and isolation of a malfunction or a mistake.

Digit: One of the symbols 0, 1, ..., 9, used to designate a quantity smaller than n for a base-n number system.

Directory: An auxiliary list for which each entry corresponds to one sublist in the object list; also an auxiliary list for searching a lower level directory and for finding a subdirectory there.

Direct Access: Retrieval or storage of data by providing a reference to its physical location on a volume.

Displacement: The difference (in bytes) between the contents of a base register (or the address represented by a symbol) and a referenced storage location.

Dummy: Pertaining to the characteristic of having the appearance of a specified thing but not having the capacity to function as such.

Edit: To modify the form or format of data; e.g., to insert or delete characters such as page numbers or decimal points.

Effective Address: The absolute address of the current operand. This may differ from that of the instruction in storage.
Error: 1. A general term to indicate that a data value is not correct or that a machine component is malfunctioning.

2. A specific term for the amount of loss in precision.

Execute: To carry out an instruction or perform a routine.

Explicit Addressing: Specification of an address by a base register and a displacement in the form D(B).

Expression: A symbol or self-defining term used in the operand of a statement.

Extent: Area of a disk file specified by an upper limit and a lower limit.

Fetch (program): 1. To obtain requested load modules and load them into main storage, relocating them as necessary.

2. A control routine that accomplishes: 1 holds to represent data and a card as in 1 before being punched.

File: A collection of related records treated as a unit, e.g., in inventory control, one line of an invoice forms an item, a complete invoice forms a record, and the complete set of such records forms a file.

File Label: Label containing information applicable to a given data file or portion of a data file stored on a particular volume.

File Reorganization: A term used to describe the process of writing a new file from an indexed sequential file, purging records that are tagged for deletion, and their sequential positions in the prime data area.

Fixed-Length Record: A record having the same length as all other records with which it is logically or physically associated.

Flow Chart: A graphical representation for the definition, analysis, or solution of a problem in which symbols are used to represent operations, data, flow and equipment.

Garbage Collection: Collection of cells deleted by linked lists operation to make available again for use.

Hardware: The mechanical magnetic, electrical and electronic devices or components of a resident system from card input.

Hierarchy: An information source which can be represented by an aborescence.
Hopper: A device that holds cards and makes them available to a card feed mechanism. Synonymous with input magazine. Contrast with card stacker.

Identification: A code number or code name which uniquely identifies a record, block, file or other unit of information.

Image: An exact logical duplicate stored in a different medium.

Immediate Address: The designation of an instruction address which is used as data by the instruction of which it is a part.

Implied Address: The address assigned to a symbol by the basic assembler program.

Index Register: A register whose content is added to or subtracted from the operand address prior to or during the execution of an instruction.

Indexing: A technique of address modification often implemented by means of index registers.

Indirect Pointer: A pointer which depicts the place a file pointer can be obtained.

Information Structure: Inherent property of the information in a given set of data. This property may be by design or by the natural way the data appear in the given set of data.

Initialize: To set certain counters, switches and addresses at specified times in a computer routine.

Initial Program Loader: A Control program, available in a card and a disk version; loads basic monitor into core storage; used to assign actual input/output addresses to symbolic addresses SYSRDR and/or SYSRES; places name of Job Control program into communication region of basic Monitor; required for the initialization of the card-resident or disk-resident system.

Input:
1. The data to be processed.
2. The state of sequence of states occurring on a specified input channel.
3. The device or collective set of devices used for bringing data into another device.
4. A channel for impressing a state on a device or logic element.
5. The process of transferring data from an external storage to an internal storage.
6. Pertaining to any entities such as are quoted above.

Input/Output:

1. Common abbreviation I/O. A general term for the equipment used to communicate with a computer.

2. The data involved in such communication.

3. The media carrying the data for Input/Output.

Input Area: The area of internal storage into which data is transferred from external storage.

Installation: A general term for a particular computing system in the context of the overall function it serves and the individuals who manage it, operate it, apply it to problems, service it, and use the results it produces.

Instruction: A statement that specifies an operation and the values of locations of all operands. In this context, the term instruction is preferable to the terms command or order which are sometimes used as synonyms. Command should be reserved for electronic signals. Order should be reserved for sequence, interpolation and related usage.

Instruction Format: The allocation of bits or characters of a machine instruction to specific functions.

Interpreter: Translator.

Interrupt: A break in the normal flow of a system or routine such that the flow can be reused from that point at a later time.

Inverted and File: A file that consists of one sub-file for each value a key may have and one inverted file corresponding to each key field in the record for which a search may be necessary.

Job Control Program: A control program, resides in core storage between jobs and provides for automatic job-to-job transition. Performs I/O device assignment. It causes the Basic Monitor to load the next program.

Job Control Statement: Any one of the control statements in the input stream that identifies a job or defines its requirements and options.

Label:
1. A physical identification record on magnetic tape located either preceding or following a data file, or both. If a data file extends beyond a single reel of tape, a label can be placed preceding and following the data on each reel.
2. A physical identification record disk which identifies the volume or file.

Language: A defined set of characters which are used to form symbols, words, etc., and the rules for combining these into meaningful communications i.e., English, French, Algol, Fortran, COBOL, etc.

Language Translator: A general term for any assembler, compiler, or other routine statements in one language which produces equivalent statements in another language.

Library: A group of files.

Library Management Programs: Collective term for four service programs; core-image maintenance, macro maintenance, directory service, and allocation organization programs.

Link: The pointer filled for a linked list.

Linkage: The interconnections between a main routine and a closed routine from the main routine.

Linkage Editor: A service program. It relocates programs or phrases and links separately assembled programs or phrases.

Load:
1. To fetch, i.e., to read a load module into main storage preparatory to executing it.
2. To place data into internal storage.

Load Program: A service program that, unlike the other service programs, is contained in punched cards. It creates a disk.

Load Module: The output of the linkage editor; a program in a format suitable for loading into main storage for execution.

Location: A position in storage that is usually identified by an address.

Logical Record: A record identified from the standpoint of its content, function, and use rather than its physical attributes. It is meaningful with respect to the information it contains. (Contrasted with physical record.)

Logical Unit Table: A feature of the Basic Monitor. It has twenty-six logical unit blocks, each of which refers to one specific symbolic I/O address. These symbolic addresses are related to physical I/O device addresses by means of ASSIGN control cards.
Loop: A sequence of instructions that is repeated until a terminal condition occurs.

Machine Address: Same as absolute address.

Machine Code: Same as operation code.

Machine Instructions: An instruction that the particular machine can recognize and execute.

Machine Language: A language that is used directly by a given machine.

Macro Instruction: An instruction that is replaced in a routine by a predetermined sequence of machine instructions.

Macro Library: A disk area containing the macro definitions required by the macro instructions issued in user-written programs. Contains source statements to generate commonly used routines.

Macro Maintenance Program: A service program. It updates the macro library and directory. It is used to add and/or delete macro definitions.

Magnetic Tape: Ink containing particles of magnetic substance which can be detected or read by automatic devices e.g., the ink used for printing on some bank checks for magnetic ink character recognition.

A tape with a magnetic surface on which data can be stored. A tape of magnetic surface used as the constituent in forms of magnetic codes.

Mask: An alphameric character string consisting of one or more digits, used to test or alter the contents of storage positions.

Mnemonic Code: A technique to assist the human memory. A mnemonic code enables the original word and is usually easy to remember, e.g., MPY for multiply and ACC for accumulator.

Module: (programming): The input to, or output from, a single execution of an assembler, compiler, or linkage editor; a source, object, or load module; hence, a program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading.

Multi-Extent Disk File: File stored on a disk pack in several areas or defined by more than one extent.

Multilist: A linked list which may contain shared sublists.

Multi-Pack Disk File: File stored on more than one disk pack.
Multi-Volume Disk File: The same as Multi-Pack Disk File.

Name: An alphameric character string, normally used to identify a program.

Object Module: The output of a single execution of an assembler or compiler, which constitutes input to linkage editor. An object module consists of one or more control sections in relocatable, though not executable, form and an associated control dictionary.

Operation:
1. The act specified by a single computer instruction.

2. A program step undertaken or executed by a computer, e.g., addition, multiplication, extraction, comparison, shift or transfer. The operation is usually specified by the operation part of an instruction.

Operation Code: The code that represents the specific operations of a computer.

Output:
1. Data that has been processed.

2. The state or sequence of states occurring on a specified output of a device.

3. The device or collective set of devices used for taking data out of a device.

4. A channel for expressing a state on a device or logic element.

5. The process of transferring data from an internal storage to an external storage.

6. Pertaining to any entities such as are quoted above.

Output Area: The area of internal storage from which data is transferred to external storage.

Overflow: That portion of data that exceeds the capacity or the allocated unit of storage, pertaining to the generation of overflow as in Overflow Area.

Overflow Area: The area a load module or a segment of a load module used for addition of records to a list or sublist which has no room left in it.

Overlay: To place a load module or segment of a load module into main storage locations occupied by another load module or segment which has already been processed.
Pack: To combine two or more units of information into a single physical unit to conserve storage.

Packed Decimal: Storage technique whereby two digits or one digit and sign are stored per byte.

Padding: A technique used to fill out a block of information with dummy records, words or characters.

Phases: The smallest addressable unit in core-image library of a disk-resident system.

Physical and Logical Unit Tables Service Program: A service program. This program (FSERV) is used to display, and/or change the permanent device assignments, and/or to change the configuration byte of the Basic Monitor on the system disk pack.

Physical Record: A record identified from the standpoint of the manner or form in which it is stored and retrieved, that is, one that is meaningful with respect to access. (Contrasted with Logical Record.)

Physical Unit Table: A feature of the Basic Monitor. It has eight physical blocks, each of which contains a physical device address. Pointers to these entries are inserted into the logical unit table by means of ASSGN control cards.

Printer: A device which expresses coded characters as hard copy.

Problem Program: Any of the class of routines that perform processing of the type for which a computing system is intended, and including routines that solve problems, monitor and control industrial processes, sort and merge records, perform computations, process transactions against stored records, etc.

Processing Program: A general term for any program that is not a control program.

Program:
1. The plan for the solution of a problem including data gathering, processing and reporting.
2. A group of related routines which solve a given problem.

Programming Language: A language used to prepare computer programs.

Pseudo-Register: A register with fixed contents used in conjunction with an IBM System/360.

Punched Card: A card punched with a pattern.
Read: To transfer information from an input device to internal or auxiliary storage.

Reader: A device which converts information in one form of storage to information in another form of storage.

Record: A general term for any unit of data that is distance from all others when considered in a particular context.

Register: A device capable of storing a specified amount of data such as one half word.

Relative Address: An address expressed by a previously defined symbol and a displacement. (e.g., FID-10)

Relocatable Area: An area on the system disk pack to temporarily hold an object module, thus permitting the assembly or compilation and the execution of a program or program phase in one job.

Relocate: In programming, to move a routine from one portion of internal storage to another and to automatically adjust the necessary address references so that the routine, in its new location, can be executed.

Report Generator and Report Program Generator (RPG): A program which constructs reports or report-writing programs in accordance with input specifications of the data file and of the desired report.

Reset:
1. To restore a storage device to prescribed initial state, not necessarily that denoting zeros.
2. To place a binary cell into the zero state.

Restart: To return to a previous point in a program and resume operation from that point.

Search Key: A key used to find a record which has a presearched identity within a file.

Seek: To position the access mechanism of a direct-access device at a specified location.

Self-Defining Term: A term with an implied value (e.g., 300, X"2A", C"F").

Semantics: The meanings which govern subsequent interpretation.

Service Programs: A collective term used to refer to the Library Management Program.

Single Extent Disk File: A file stored on disk where the file has exactly one extent.
Sort: The process by which a data set of logical records is sequenced according to the collating-sequence value of the control field of the records. Also a program that performs the process.

Source Language: A language that is an input to a given translation process.

Source Program: A program written in a source language.

Special Characters: In a character set, a character that is neither a numeral nor a letter, e.g., *, -, $ and blank.

Statement: In computer programming, a meaningful expression or generalized instruction in a source language.

Step:
1. One instruction in a computer routine.
2. To cause a computer to execute one instruction.

Store:
1. To enter data into a storage device.
2. To retain data in a storage device.

Storage:
1. Pertaining to a device into which data can be entered and from which it can be retrieved at a later time.
2. Loosely, any device that can store data.

Storage Allocation: The assignment of blocks of data to specified blocks of storage.

Storage Capacity: The amount of data (in bytes) that can be contained in a storage device.

Subroutine: A routine that can be part of another routine.

Switch:
1. A symbol used to indicate a branching point, or a set of instructions to condition a branch.
2. A physical device which can alter flow.

Symbol Table: A mapping for a set of symbols to another set of symbols or numbers.

Symbolic Address: An address expressed in symbols convenient to the programmer.
Symbolic I/O Device (e.g., SYRES, SYSIPP, SYS005): This address is related to an actual address by means of the logical unit table.

Symbolic Language: The discipline that treats formal logic by means of a formalized language or symbolic calculus whose purpose is to avoid the ambiguities and logical inadequacies of natural language. Advantages of the symbolic method are greater exactness of formulation and greater exactness of formulation and power to deal with complex material.

Syntax: The rules for constructing admissible combinations of the characters in the basic alphabet.

System:
1. A collection of consecutive operations and procedures required to accomplish a specific objective.
2. An assembly of objects united to form a functional unit.

System Directory: A table on the system disk pack listing the addresses and sizes of the core-image library directory, the macro library and the directory, and the locatable area.

System Disk Pack: The disk pack on which the user's disk-resident system is located.

Table: A collection of data, each item being uniquely identified either by some label or by its relative position.

Table Look-Up: A procedure for obtaining the function value corresponding to an argument from a table of function values.

Throughput: A measure of system efficiency, the rate at which work can be handled by the computing system.

Truncate: To cut off a specified spot (as contrasted with round or pad).

Unpack: To recover the original data from packed data.

User: Anyone who requires the services of a computing system.

Volume: That portion of a single unit of storage media that is accessible to a single read/write mechanism. For example, a reel of magnetic tape for a 2415 magnetic tape drive, or one 1316 Disk Pack for an IBM 2311 Disk Storage Drive.

Volume Label: Label which uniquely identifies the volume.

Volume Table of Contents (VOTC): A table associated with a disk volume, which describes each data set on the volume.
Zero: The elimination of non-significant zeros in a number.

Zone: The 12, 11 or 0 punches in IBM card code.
I. Books, Pamphlets, and Government Publications


II. Journal Articles

Automatic Coding (Proceedings of the Symposium of Automatic Coding held
January 24-25, 1957 at the Franklin Institute in Philadelphia).  
*Journal of Franklin Institute, Monograph No. 3. Philadelphia, Pa., April, 1957.*


III. Unpublished Materials

INDEX

ADP (Automatic Data Processing), 8
Antelope Valley Construction Co., 72
Assembly Level, 28
Authorization For Work, 21
Automacity, 5
Array Search, 184

Background, 9
Bill of Material, 16
Binary Search, 159
Buffering, 183

Calculated Address Sort, 143
Calculation Specifications, 237
Canned Programs, 39
Capacity Planning, 37
Chaining, 309
Chaining Concept, 34
Computed Go To, 202
Compound Interest Rate, 197
Construction, 47
Construction Flow Chart, 50
Construction Program, 55
Contracts, 17
Cost of Manpower, 24
Cost Savings, 12
Curriculum Committee of the Association for Computing Machinery, 80

Data Base, 25
Data Structures, 79
Date Card, 235
Deque, 88
Derivative, 203
Dewey Decimal System, 101
Direct Costs, 4
Do Loop, 179
Double Step-Down, 154
Doubly Linked Lists, 96
Dynamic Storage Allocation, 163

Engineering, 26
Environment, 12
Existing System, 13
Express Formation, 178

Fabrication Level, 28
Files, 117
File Layout, 33
File Extension Specifications, 271
Forecast of Sales, 19
Fortran, 172
Functional Criteria, 7

Go To Statement, 211

Hardware Costs, 38
Hierarchy of Operation, 178
Hierarchical Ring Structure, 104
Hierarchical Structures, 103

IBM, 6
Indexed Sequential File, 305
Indirect Costs, 3
Information, Single Flow of, 5
Input Specifications, 230
Inventory, 4
Inventory Control, 11
Inventory Control Subsystem, 29
Inverted Files, 122
Installment Note, 187
Integration, 199
Integrated Data Processing, 10
Item Master, 34

Job Order (Master Job Order), 20

Knuth, 87
Linear Equation, 201
Linked Lists, 91
List Formats, 87
List of Available Space, 89
Logical Operations, 125

Material Control, 11
Merging, 148
MJO, 15
Modular Program Design, 36
Mr. Fictitious, 25
Multiple File, 132
Multiple Card Layout, 227

Network, 47

Oceano-Graphic Corporation, 3
Objectives, 7
Oldmaster, 229
Open Job Operation Detail, 34
Open Job Order, 28
Open Job Order Summary, 34
Open Purchase Requisitions, 34
Operational Gains, 12

Pegged Requirements, 34
PERT, 13
Plotting, 198
Polynomial, 204
Preorder, 108
Pre-Billing Calculation, 222
Primary Flow of Information, 26
Print Error Test, 286
Priority Rules, 31
Procedure Flow Chart, 35
Procedural Analysis, 12
Production Function, 7
Production Model, 30
Production System Proposal, 3
Product Structure File, 34
Proposal System, 24
Purchasing, 26
Purchasing Functions, 10
Purchase Master, 34

Quadratic Equation, 200
Quadratic Sort, 142
Quicksort, 140

Random Organization, 169
Registers, 62
Requirements Planning, 29
Research and Development, 17
RPG, 219

Sales, 17
Scope, 8
Search, 158
Sequential Disk Files, 296
Sequential File, 299
Shortcuts, 182
Software Costs, 39
Sort, 63
Sorting, 136
Stack, 88
Stacker Selection, 226
Standardized Records, 32
Standard Parts, 16
Standard Routing, 28
Standardization, 36
Statistics, 188
Suborganizations, 9
Subsystems, 15
Summary Punching, 276
Swanson Study, 2
System Overview, 26

Tape Cassette, 47
Tool Master, 34
Topological Sort, 141
Trees, 101

Virtual Memory, 166

Work Order, 18
Work Center Load, 28
Work Center Master, 34